

LIVERPOOL

SOFTWARE GAZETTE

OCTOBER 1980

SIXTH EDITION 75p



Final

MICRODIGITAL BARGAINS

Books are not subject to V.A.T.
If we are sold out your money will be refunded by return.
All these offers are subject to availability.

Hardware

MEMORY

Guaranteed quality components 8 x 4116
 (16k bytes)

Suitable Nascom, Apple, TRS 80, Sorcerer etc.

Price	Nett	Vat	Total
Eight chips	20.00	3.00	23.00

2 x 2114 (1K Byte)

Suitable Atom, ZX80, Superboard etc.

Price	Nett	Vat	Total
	4.00	0.60	4.60

Nascom ROM/EPROM Board

Room for 8x2708 + 8K BASIC ROM

	Nett	Vat	Total
Assembled Price	30.00	4.50	34.50

Veroframe

Suitable for Nascom

Price	Nett	Vat	Total
	20.00	3.00	23.00

Byte Books

£2 each - all 6 for £10

BASEX

A simple language and compiler for 8080 systems.

LINK 68

An M6800 linking loader

SUPERWUMPUS

A game in 6800 code or BASIC

PROGRAM DESIGN

Volume 1 of Programming Techniques

BAR CODE LOADER

For 6800, 6502, 8080 or Z 80

RA6800 ML

An M6800 relocatable macro assembler.

Books

Introduction to Microprocessors and Computing

Educational Data and Technical Services
 £1.50

Conference Proceedings of the West Coast Computer Faire

first
 second £5 each
 third
 fourth

Practical Microcomputer Programming

By W J Weller
 The 6800 £12 each
 The 8080

Nascom Programs + Information
 by Merseyside Nascom Users Group £2



Retail Premises at:
 25 BRUNSWICK STREET,
 LIVERPOOL L2 0PJ.
 Tel:- 051-227 2535/6/7

Mail Orders to:
 MICRODIGITAL LIMITED
 FREEPOST (No stamp required)
 Liverpool L2 2AB
 24 Hour - 7 Day
 ANSWERPHONE SERVICE
 on 051-236 0707

CONTENTS

Page

- | | |
|---|-------------------------------------|
| 6. 'Warning' Prolonged Use Of Pascal may seriously damage your mental health. | 36. A Readers Contribution |
| 9. This space costs just £95.00! | 40. Letters to the Editor |
| 10. A useful Pascal Program #1 | 45. Alarming your Computer |
| 14. Integer Pascal for a Nascom | 48. Apple Pips |
| 18. Structured and not so structured Programming | 50. The Romplus and Keyboard Filter |
| 20. TCL Pascal | 54. Stargate Unlocked |
| 26. A useful Pascal Program #2 | 56. Tangerine Article |
| 27. Free Classified Ads! | 58. MPL Language |
| 28. Pets Corner | 60. Nascom Notes |
| | 66. A number Processor on the Acorn |
| | 74. C.R.A. |
| | 76. ETC |

LIVERPOOL SOFTWARE GAZETTE

Editor.

0.0

The trend in this issue is towards a larger number of smaller articles with Pascal as a theme in several of them. Overall the Gazette is still a little over Apple/Pet oriented — do the users of other systems have nothing worth writing about?

0.1

With present day technology it would be feasible to produce a car with a top speed of 200 m.p.h.

What the public demand, and the manufacturers supply however, are vehicles with half this speed. What is far more important is economy, versatility, ease of use and practicality. There is a demand for the very powerful car but it is only a small part of the overall market place.

0.2

The advent of the Z-80 Softcard for the Apple II must be marked as a major effort from Microsoft, Digital Research and California Computer Systems — the card allows Apple to run Z-80 software, notably CP/M. As a result Apple can now run the vast amount of CP/M software that exists in the world and which includes such goodies as COBOLM CORAL and BASIC compilers. A further result will be an enormous boost in the use of CP/M as Apple is now the worlds best selling computer.

Shortly it will be impractical to own an Apple without at Z-80 Softcard as CP/M now seems destined to become a true industry standard.

0.3

The microcomputer media recently contained a fair amount of controversy over the language Pascal — some of it ill informed. In this issue we present a number of view points which seem to cover the full spectrum of opinion.

If any readers have arguments they would like to add to the debate any letters or articles on Pascal or structured programming in general will be gratefully received.

Publisher

1.0

A half page ad in this magazine costs £52.00 and the advertiser receives 50 free copies worth £37.50. The nett cost for a half page ad in 5,000 gazettes is thus £14.50. This has to be a bargain.

B. Everiss
BRUCE EVERISS

1st GRADE EPROMS

Through our buying we can offer the following:

2716-3 RAIL - £16.00

2716-Single 5V-£10.00

2708-Single 5V-£4.25

including VAT and Postage

COMPETITIVE VIDEO ELECTRONIC GAMES

112, Villers Ave.,
Surbutan,
Surrey.



MICRODIGITAL

25 Brunswick Street, Liverpool L2 0BJ

Tel: 051-226 0707 (24 hour Mail Order)

051-227 2535 (All other Depts.)

Mail orders to: MICRODIGITAL LIMITED,
FREEPOST (No Stamp Required) Liverpool L2 2AB

CRYSTAL ELECTRONICS CC ELECTRONICS

SHARP MZ80K

For the latest competitive PRICE

Contact us

Before you accept discounts elsewhere.

GIVE US A TRY

**CRYSTAL ELECTRONICS is the home of XTAL BASIC
ACCLAIMED BY MANY**

We KNOW the SHARP computers, we BACK the SHARP computers
What we give FREE is worth more than money

**MZ80K owners—are you XTAL followers?
NO! Then please read on.
XTAL BASIC (SHARP)**

Takes 5K less memory, has all the features of SHARP BASIC PLUS Multi dim strings, error trapping, logical operators, machine code monitor, more flexible peripheral handling, improved screen control, increased list control, auto run, if, then... else—and it doesn't stop there—it grows. You can extend the commands and functions at will—10K, 12K, 16K, BASIC7.

SHARP to XTAL BASIC conversion program is included.
£40 plus VAT (Disc version on its way)

**DESIGNERS OF MICROCOMPUTER SYSTEMS + XTAL BASIC
IS WORTH CONSIDERING ON COST ALONE.**

Members of Computer Retailers Association & Apple Dealers Association

Shop open 0930-1730 except Saturday & Sunday

40 Magdalene Road, Torquay, Devon, England, Tel: 0803 22699

Telex 42507 XTAL G

Access and Barclaycard welcome.



LIVERPOOL SOFTWARE GAZETTE

Editor/Publisher: Bruce Everiss.

Editorial Assistant: Stephanie Nowell.

Contributing Editors: J. Stout, Dr. Martin Beer, Dave Straker.

Advertising/Subscriptions: Stephanie Nowell.

Artwork: Peter Cross, John Burrows.

ADVERTISING:

Full page (17.5cm x 24cm) £95.00

Half page (Upright 8.5cm x 24cm) £52.00

Half page (Landscape 17.5cm x 11.75cm) £52.00

Agency discount 10%

Please contact Stephanie Nowell on 051-227 2535 if you would like further details.

DISCLAIMER: 'All the information in the magazine has been thoroughly debugged and tested. However, no guarantees are made as to its truth or validity'.

TRADE DISTRIBUTION:— Computer Bookshop, 43-45 Temple Street, Birmingham, 021-643 4577.

(c) LIVERPOOL SOFTWARE GAZETTE 1980

THE LIVERPOOL SOFTWARE GAZETTE is published bi monthly by Liverpool Software Gazette Limited, 14 Castle Street, Liverpool L2 0TA. Registered in England No 1477285

Subscriptions: Within Great Britain. £9.00 for 12 issues. Individual copies by post 75p. Please tell us the issue you would like to start a subscription with!

REPRINTS: Articles that are explicitly marked as having restricted reproduction rights may not be copied or reprinted without written permission from Liverpool Software Gazette. All other articles may be reprinted for any non-commercial purpose provided a credit line is included stating that said material was reprinted from the Liverpool Software Gazette, 14 Castle Street, Liverpool. L2 0TA. Please send copies of any reprints to Liverpool Software Gazette, attention of Bruce Everiss.

WARNING!

PASCAL



Prolonged use of Pascal
may seriously damage
your mental health.

by Raymond Anderson

Pascal is being heavily pushed by many people, at present. This article aims to show that Pascal should not be heavily invested in because it will start to crack just when the programmer needs most support from a language.

Before discussing features of Pascal which cause problems to the serious programmer tackling real problems, or trying to provide tools for other people, I will briefly outline my experience with languages.

I have not used Pascal much, despite learning it out of interest, because I have always had a better tool available for the problem in hand. The languages I use mostly are Algol 68, BABBAGE, various assemblers, BASIC, and text editors such as ZED. The machines I use are Z-80, GEC4082, IBM370 and PDP11, and I have a fairly wide experience of using LISP, COBOL, FORTRANS 4 and 77, ALGOL-W, BCPL, C, SNOBOL, CORAL 66, PL/1 and others with which to compare Pascal.

People learning Pascal after using BASIC for some time soon realise the advantages a structured language can give them, and become proud of their mastery of Pascal and keen to castigate people who still use BASIC. Unfortunately, Pascal is far from ideal, and by the time people have realised this they have to plod on as best they can to ensure their investment in old software is not lost — just as the FORTRANners have done for 20 years.

My viewpoint is that it is better to learn a language which allows expansion within itself as much as possible, to old investment is not lost by providing more facilities, and portability is ensured. Algol 68 is my personal choice, although ADA seems to have achieved this, and in their own ways so have APL, LISP and FORTH.

Software houses push Pascal because the compiler is fairly easy to write and easy to sell to somebody unfamiliar with the language.

Pascal programmers push Pascal so more users will put pressure on software houses to sell better compilers, and compilers for more machines.

Some teachers push Pascal because it is quite easy to teach, and their pupils will prefer a trandy language.

Magazines feature Pascal articles because the people above want to write about it. (Although I'm not sure that the readers enjoy them).

Algol 68 programmers worry that a lot of people will get their fingers burnt on Pascal, and lose faith in their climb up the Algol tree.

The following sections each describe a particular part of Pascal which I consider badly designed. Before I delve more deeply I would like to point out that Pascal does have some rather interesting features, such as 'subranges', 'sets' and 'packed records' which I feel are good points, and many features which are common nowadays, and very useful:

records, recursion, datatypes and so on.

Pascal is useful as a first step towards structured programming, but you must not wait too long on the first step or you will grow too weak to climb further.

Procedure Libraries

Programmers in an applications environment find it useful if they can use a set of general purpose routines provided by experts in some field. Examples of such sets are the GINO and GINOGRAPH graphics routines and the NAG subroutine library for FORTRAN and Algol 68. There are also many cases where facilities are embedded in languages to make certain activities easier — for example handling databases or laboratory equipment. Unfortunately, such useful facilities are very difficult to provide in Pascal. This is because a procedure which takes in array as a parameter can only take arrays of size fixed in the procedure declaration. A procedure written for integer arrays of 100 elements could not be used with arrays of 101, 99 or 3 elements!

Even FORTRAN provides better facilities for making libraries available — indeed this is one of its main strengths in engineering and scientific fields. Algol 68 allows new operator definitions for any data types. APL users to business often use a set of functions provided by programmers so they do not need to concern themselves with detail.

This serious fault in Pascal was caused by an obsessive desire to simplify storage allocation, but the drawbacks are far too severe to justify this simplification.

Input/Output

When a program has been written and the algorithms checked using test data, it often has to be used by untrained users and on real data.

Users will often feed in corrupt data — letters instead of numbers, blank lines, overlarge numbers or just gibberish, and a good program will give some indication of the error and try to continue to find more errors, rather than just bombing out.

Writing a robust program to simply read a number from a file is very difficult in Pascal, because there are so many faults that can occur which the program cannot sort out by using the inbuilt routines to read an integer.

Consider some of the errors that could occur:

- (i) The file is empty
- (ii) The file is a line of blanks
- (iii) There are blank lines before the number
- (iv) The file starts with the letter X
- (v) There are superfluous characters after the number
- (vi) The number is too large or small to hold an integer value

The Pascal programmer has to forget the system routines, and write his own new ones to read character by character. More room for error and more work.

Algol 68, COBOL, and PL/I allow checking on input, and FORTRAN 77 has a method of trapping errors.

Pascal has no facilities for opening and creating new files, and cannot enquire about the status of files it is connected to except for detecting end of line (EOLN) and end of file (EOF).

It is in fact so inadequate at file handling that almost all implementations have their own (non-standard) extensions, thereby reducing portability.

Static Array Bounds

When Pascal was designed, it was decided that several restrictions would be made to simplify compilation, and increase efficiency on the processors of the early 70's. One of these restrictions is that the bounds of arrays had to be constants available at compile time.

This means that a program or routine cannot declare the workspace it needs at a particular time. Instead, it has to declare an array which is as large as it should ever need every time, thus wasting space on most occasions.

On microprocessors with a limited address space, a waste of space is worse than a slight reduction in speed.

Declarations

Pascal requires that declarations of all procedures, variables, and so on are made at the top of the main program body, and at the top of each procedure body.

This is another example of how a restriction meant to make life slightly easier for the compiler writer is a pain for the programmer, especially when he is working on large programs. Once again, the result is a waste of space.

Suppose one block in the program uses some integers, and another uses some reals, and these variables are confined to their respective blocks. If the variables could be declared in their respective blocks, there would be re-use of store (saving space), no access to the variables outside their block (preventing errors), and the type of the variables is clear near where they are used. There is no run-time penalty, and hardly any increase in compiler complexity.

Procedures should be used in place of the blocks, but this can lead to a proliferation of small (25 line) procedures, and marked loss of speed.

The Loop Construction

The Pascal FOR/DO loop is very restricted and often has to be replaced by some open coded equivalent using a WHILE construct.

There is no 'STEP' or 'BY' part, so the loop variable can only go up or down in steps of one. This petty restriction, supposedly for 'efficiency' is another place Pascal hinders straightforward coding and obscures intentions.

The loops may as well always start from 1!

A WHILE part cannot be combined with the DO loop, so an exit from the loop on some condition (for example the end of a search) has to be by a 'GOTO' to some label outside.

The final oddity is that the loop variable has to be declared to the normal manner at the top of the program or procedure. Apart from slowing down coding, this has the undesirable effect of making the variable accessible to code in the loop or in procedures declared 'below it in scope' where it can be accidentally altered, causing chaos.

The CASE Construction

A useful feature of Pascal is its CASE construction, allowing the selection of one of several courses of action depending on the value of a variable, or which member of a set it is. Unfortunately, this feature is marred by the lack of a DEFAULT or OUT part which is a way of specifying what to do if there is no relevant CASE action. Most Pascal implementations do nothing if the selector is not catered for, but one usually wants to take some action if an exception arises, and one has to use an IF construction of some sort. This is yet another serious flaw in Pascal, which this time has no excuse.

Bit Handling

Pascal does not provide access to the individual bits in integers to allow packing of data, logical operations or shifts, often very important for systems programming.

Once again many implementations have added extra operators and inbuilt data types but these are extensions and reduce portability.

Insecurities

Pascal is a rather more secure language than (say) FORTRAN, which means that not so many expensive run-time checks would be required behind the scenes to prevent accidental or deliberate but dangerous tampering with data.

Unfortunately, Pascal becomes insecure in the areas which many programmers are already having difficulty thinking about what should be happening if all goes well, the areas of variant, pointers, and heap storage.

Firstly, the mark and release scheme of storage management is completely unchecked, and allows easy corruption of store. Admittedly a garbage collector is a complex beast, but it saves the programmer a lot of work in storage management, one of the major parts of many problems.

Secondly, a variant record can be established with one type, and then assignments can be made to the fields of the record pretending it is some other type, possibly one which occupies more store, thereby corrupting neighbour data.

Thirdly, since a procedure parameter does not have a 'type' associated with it, checking of the number of arguments used by the procedure when it is called inside another procedure are highly difficult (perhaps impossible) at compile time, and usually never made at compile time. This is a similar problem to that encountered in FORTRAN, where it is impossible to tell at compile time if subprograms are being called with the correct number and type of arguments.

Summary

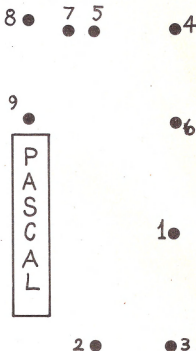
I have shown that Pascal has deficiencies in many areas, ranging from annoying omissions (STEP in DO-loop, OUT in CASE construct) to serious design flaws (fixed size of array parameters).

These deficiencies do not mean that the language is unusable, but they mean that difficulties arise just when the compiler and language should be helping most; in large programs and structured design of programs.

The insecurities mentioned in the final section mean that as things get harder and harder to keep track of, data corruptions can occur causing faults in unexpected places, and storage gets harder to manage.

By all means use Pascal if you need a structured language quickly, but keep looking around for a longer lasting tool and invest in it as soon as you can.

FIG. 1 JOIN THE DOTS



After reading each section, join the next dot in the figure above. This will help you come to a deeper understanding of the article.

PASCAL



THIS SPACE COSTS JUST

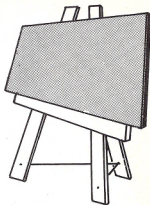
£95.00

**AND YOU
GET 50 COPIES
OF THE SOFTWARE GAZETTE
FREE!**

To book your advertisement
in Britains most talked-about
Computer Magazine –
Phone Steph on
051-227 2535
Copy date for December issue is
November 21st

A USEFUL PASCAL PROGRAM!

#1 *David M. Evans*



```
(* $L PRINTER: *)
PROGRAM FILEDUMP;
```

```
(* PROGRAM TO PRINT OUT FILES IN HEXADECIMAL
AND CHARACTER FORMAT.
```

```
CONST TWENTYFOUR = 24;
      EIGHT = 8;
```

```
(* PRINT CONTROL CONSTANTS *)
```

```
C8P3 = 28; (*8.3 P/I*)
C10 = 29; (*10 C.P. I*)
C12 = 30; (*12 C.P. I*)
C16P5 = 31; (*16.5 C.P. I*)
LF = 10; (*LINE FEED *)
FF = 12; (*PAGE*)
DOUBLE = 1; (*DOUBLE WIDTH*)
SINGLE = 2; (*SINGLE WIDTH*)
```

```
(* END OF PRINT CONTROL CHARS*)
```

```
TYPE BLK = PACKED ARRAY [0..511] OF CHAR;
      RETYPE = (OK, BAD);
```

```
VAR PR : TEXT;
    WAITINDICATOR, OUTDEVICE : CHAR;
    VOL, FILENAME, NARRATIVE : STRING;
    LINEENO, BYTEPOS, BLOCKCNT, LAST, START :
    INTEGER;
    LINE : PACKED ARRAY [1..76] OF CHAR;
    RESULT : RESTYPE;
    BLOCK : BLK;
    CHAREQUIV : PACKED ARRAY [0..15] OF
    CHAR;
    INFILE : FILE;
    R : INTEGER;
    FLIP : BOOLEAN; (* PAGE SKIP IND. *)
```

```
PROCEDURE INIT;
VAR X : INTEGER;
BEGIN
```

```
    (* SET UP CHARACTER EQUIVALENTS OF
    HEXADECIMAL DIGITS 0 TO F *)
```

```
    FOR X := 0 TO 9 DO (* 0 TO 9 *)
    CHAREQUIV[X] := CHR (48+X);
```

```
    FOR X := 10 TO 15 DO (* A TO F *)
    CHAREQUIV[X] := CHR (65-10+X);
```

```
    REPEAT
    WRITELN('DO YOU WANT TO SEND THE O/P TO');
```

```
    WRITELN('SCREEN OR PRINTER ? (S/P) ');
    READ( OUTDEVICE); WRITELN;
```

```
    UNTIL OUTDEVICE IN ['S', 'P'];
```

```
    IF OUTDEVICE = 'S' THEN
    REWRITE(PR, 'CONSOLE:')
```

```
    ELSE
    REWRITE(PR, 'PRINTER:');
```

```
    FLIP := TRUE; (*PAGE SKIP ON FIRST BLOCK*)
    WRITELN('ENTER A LINE OF NARRATIVE FOR PRINT
    HEADING'); READLN( NARRATIVE);
```

```
    IF OUTDEVICE = 'P' THEN
    BEGIN
```

```
        REPEAT
        WRITELN(' DO YOU WANT WAITS INSERTED, ');
        WRITELN('FOR PRINTER TO CATCH UP? (Y/N)');
        READ( WAITINDICATOR); WRITELN;
```

```
    UNTIL WAITINDICATOR IN ['Y', 'N'];
    END;
```

```
    END; (* INIT *)
    PROCEDURE GETFILEPARMS;
    BEGIN
```

```

WRITELN('ENTER VOL');
READLN(VOL);

WRITELN('ENTER FILE');
READLN(FILENAME);

WRITELN('ENTER START BLOCK (0..)');
READLN(START);

WRITELN('ENTER LASTBLOCK');
READLN(LAST);

END;
PROCEDURE CHARPR(C:CHAR;CHPOS, HEXPOS:
INTEGER);
(* PUTS 1 CHAR INTO LINE AS HEX & CHAR *)
VAR T: INTEGER;
BEGIN
    T:=ORD(C) DIV 16;(*HI-ORD HALF-BYTE*)
    LINE[HEXPOS]:=CHAREQUIV[T];

    T:=ORD(C) MOD 16;(*LO-ORDER HALF-BYTE*)
    LINE[HEXPOS+1]:=CHAREQUIV[T];

    (*IF CHARACTER IS UNPRINTABLE, THEN
    PUT BLANK IN CHARACTER POSITION*)

    IF (C<CHR(32)) OR (C>CHR(126))
    THEN LINE[CHPOS]:=' '
    ELSE LINE[CHPOS]:=C;

END;
PROCEDURE WRLINE(BYTEPOS, NUMOFBYTES:
INTEGER);
(* THIS PROC WRITES A LINE OF DUMP OF
'NUMOFBYTES' LONG,
STARTING WITH BYTE 'BYTEPOS' *)
VAR X, Y: INTEGER;
    HL, CL: INTEGER; (* LINE LOCATION INDICES *)
BEGIN
    Y:=BYTEPOS;
    FILLCHAR(LINE[1], 76, ' ');(*BLANK OUT LINE*)

    HL:=1;(*HEX CHAR INDEX*)
    CL:=1;(*CHAR INDEX*)

    FOR X:=1 TO NUMOFBYTES DO
    BEGIN
        CHARPR(BLOCK[Y], 51+CL;(*LOCATION FOR
        CHAR*), HL;(*LOCATION FOR HEX*));
        CL:=CL+1;
        HL:=HL+2;
        Y:=Y+1;
    END;

    LINE[51]:='*';
    LINE[76]:='*';
    WRITELN(PR, BYTEPOS: 3, ' ', LINE);
END;
FUNCTION GETABLOCK:RESTYPE;
VAR X, Y: INTEGER;
    BEGIN
        GETABLOCK:=OK;
        (*$1-*)
        R:=BLOCKREAD(INFILE, BLOCK, 1, BLOCKCNT);

        (*$1+*)
        IF(IORESULT<>0) OR (R<>1) THEN
            GETABLOCK:=BAD;

        END;
        PROCEDURE BLOCKHDL;
        BEGIN
            WRITE(PR, CHR(C8P3));
            WRITE(PR, CHR( DOUBLE ));
            WRITELN(PR, ' ':8, 'FILE PRINT UTILITY');
            WRITE(PR, CHR( SINGLE ));
            WRITELN(PR, NARRATIVE);
            WRITELN(PR, 'VOL : ', VOL);
            WRITELN(PR, 'FILE : ', FILENAME);
            WRITELN(PR, 'BLOCK : ', BLOCKCNT);
            WRITE(PR, CHR(C10), CHR(SINGLE));

        END;
        PROCEDURE WAIT;
        (*
        WAITS A FEW SECONDS FOR THE PRINTER
        TO CATCH UP IF REQUIRED.
        (ONLY OF USE IF USING A SERIAL PRINTER)
        *)
        VAR X: INTEGER;
        BEGIN
            IF WAITINDICATOR='Y' THEN
                FOR X:=1 TO 25000 DO;

            END;
            BEGIN (*MAIN*)

                INIT;(* INITIALISE CHARACTER EQUIVALENTS
                AND GET OUTPUT LOCATION*)

                GETFILEPARMS;

                BLOCKCNT:=START;

                (*$1-*)
                RESET(INFILE, CONCAT(VOL, FILENAME));
                (*$1+*)

                IF IORESULT<>0 THEN
                    BEGIN
                        RESULT:=BAD;
                        WRITE(CHR(7));(*BEEP*)
                        WRITELN('VOL/FILE BAD - HIT<RET>');
                        READLN;

                    END
                    ELSE RESULT:=OK;

                RESULT:=GETABLOCK;

                WHILE (RESULT=OK) AND (BLOCKCNT X=LAST) DO
                    BEGIN

```

FILE PRINT UTILITY

EXAMPLE OF FILEDUMP OUTPUT

VOL : TEST4:

FILE : FILEDUMP.TEXT

BLOCK : 3

```

0 53545950453D284F4B2C424144293B0D10250D1020564152
24 2050523A544558543B0D102457414954494E444494341544F
48 522C4F55544445564943453A434841523B0D1024564F4C2C
72 46494C454E414D452C4E41525241544956453A535452494E
96 473B0D10244C494E454E4F2C42595445504F532C424C4F43
120 4B434E542C4C4153542C53544152543A494E54454745523B
144 0D10244C494E453A5041434B45442041525241595B312E2E
168 37365D204F4620434841523B0D1024524553554C543A5245
192 53545950453B0D1024424C4F434B3A424C4B3B0D10244348
216 415245515549563A5041434B45442041525241595B302E2E
240 31355D204F4620434841523B0D1024494E46494C453A4649
264 4C453B0D102453A494E54454745523B0D1024464C49503A
288 424F4F4C45414E3B282A205041474520534B495020494E44
312 2E2A290D10220D102050524F43454455524520494E49543B
336 0D102056415220583A494E54454745523B0D102042454749
360 4E0D1024282A205345542055502043484152414354455220
384 4551554956414C454E5453204F460D102748455841444543
408 494D414C20444947495453203020544F2046202A290D1024
432 0D1024464F4C5220583A3D3020544F203920444F20282A2030
456 20544F20392A290D10274348415245515549565B585D3A3D
480 4348522834382B58293B0D10240B00000000000000000000
504 0000000000000000

```

```

*STYPE=(OK,BAD); % VAR*
* PR:TEXT; %WAITINDICATOR*
* R:OUTDEVICE;CHAR; %VOL,*
* %FILENAME,NARRATIVE;STRIN*
* G: %LINENO,BYTEPOS,BLOC*
* %CNT, LAST, START; INTEGER;*
* % LINE:PACKED ARRAY[1..*
* %76] OF CHAR; %RESULT;RE*
* %TYPE; %BLOCK;BLK; %CH*
* %AREQUIV:PACKED ARRAY[0..*
* %15] OF CHAR; %INFILE;FI*
* %LE; %R;INTEGER; %FLIP;*
* %BOOLEAN;(% PAGE SKIP IN*
* ,*) " PROCEDURE INIT;*
* % VAR X:INTEGER; BEGI*
* %N %(* SET UP CHARACTER *
* %EQUIVALENTS OF 'HEXADEC*
* %IMAL DIGITS 0 TO F *) %*
* % $FOR X:=0 TO 9 DO (% *O
* % TO 9*) 'CHAREQUIV[X]:=*
* %CHR(48+X); % $
*

```

FILE PRINT UTILITY

EXAMPLE OF FILEDUMP OUTPUT

VOL : TEST4:

FILE : FILEDUMP.TEXT

BLOCK : 4

```

0 1024464F5220583A3D313020544F20313520444F20282A20
24 4120544F2046202A290D10274348415245515549565B585D
48 3A3D348522834352D31302B58293B0D10240D1024524550
72 4541540D102657524954454C4E2827444F20594F55205741
96 4E5420544F2053454E4420544845204F2F5020544F27293B
120 0D102657524954454C4E282753435245454E204F52205052
144 494E544552203F2028532F502927293B0D10265245414428
168 4F5554444556494345293B57524954454C4E3B0D1024554E
192 54494C204F555444455649434520494E205B2753272C2750
216 275D3B0D10240D10244946204F55544445564943453D2753
240 27205448454E0D10242020524557524954452850522C2743
264 4FE534FC4C453A27290D1024454C53450D10265245525249
288 54452850522C275052494E5445523A27293B0D1020202020
312 200D1024464C49503A3D545255453B20282A504147452053
336 4B4950204F4E20464952535420424C4F434B2A290D102020
360 2020200D102457524954454C4E2827454E5445522041204C
384 494E45204F46204E415252415449564520464F5220505249
408 4E542048454144494E4727293B0D1024524541444C4E284E
432 4152524154495645293B0D1020202020200D10244946204F
456 55544445564943453D275027205448454E200D1024424547
480 494E0D10285245504541540D102A57524954454C4E282744
504 4F20594F55205741

```

```

* %FOR X:=10 TO 15 DO (% *
* %A TO F *) 'CHAREQUIV[X]:
* %:=CHR(65-10+X); % $REP*
* %AT &Writeln('DO YOU WA*
* %NT TO SEND THE O/P TO');*
* % &Writeln('SCREEN OR PR*
* %INTER ? (S/P)'); &READ*
* %OUTDEVICE);Writeln; %UN*
* %TIL OUTDEVICE IN ['S','P*
* %']; % $IF OUTDEVICE='S'*
* % THEN % REWRITE(PR,'C*
* %ONSOLE'); %ELSE &REWR*
* %TE(PR,'PRINTER:'); %
* % $FLIP:=TRUE; (%PAGE S*
* %KIP ON FIRST BLOCK*) %
* % %Writeln('ENTER A L*
* %INE OF NARRATIVE FOR PRI*
* %NT HEADING'); %READLN;*
* %NARRATIVE); %IF O*
* %UTDEVICE='P' THEN %BEG*
* %IN (REPEAT &Writeln('D*
* %O YOU WA
*

```

```
(* ENSURE PAGE SKIP EVERY SECOND BLOCK*)
IF FLIP = TRUE THEN
  BEGIN
    FLIP := FALSE;
    PAGE (PR);
  END
ELSE
  FLIP := TRUE
```

```
BYTEPOS := 0
BLOCKHDS;
FOR LINENO := 1 TO 21 DO (* 21 LINES OF 24
  BYTES*)
  BEGIN
```

```
    WRLINE(BYTEPOS, TWENTYFOUR);
    BYTEPOS := BYTEPOS+TWENTYFOUR;
```

```
  END;
  WRLINE(BYTEPOS, EIGHT); (*LAST LINE IS
  ONLY EIGHT BYTES*)
```

```
  WRITE(PR, CHR(LF), CHR(LF));
  BLOCKCNT := BLOCKCNT+1;
```

```
IF OUTDEVICE = 'P' THEN WAIT;
```

```
RESULT := GETABLOCK;
```

```
END;
END.
```

SUPPORT MEMBERS OF THE COMPUTER RETAILERS ASSOCIATION . . .



THEY WILL SUPPORT YOU.

For further details on the associations aims, membership, code of conduct etc.

Please contact: Mrs Gibbons,
Owles Hall, Buntingford,
Hertfordshire, SE9 9PL.
Tel. (0763) 71209

Computer Bookshop Gives MICROBOOK Retailers

FAST, FREE DELIVERY Ex Stock!

And the best selection of MICRO-
COMPUTER books in the business

Send for full details of all our BIG
Selling Books NOW!!!

The Computer Bookshop



Temple House, 43/48 New Street,
Birmingham B2 4LA.
Telephone: 021-643 4577.

AVOLON SOFTWARE

Z80
280 EDITOR ASSEMBLER



Zen is a complete system for the Z80 assembly programmer, it's reliability proven on thousands of computers. All versions of Zen consist of a cassette, user manual and full assembly listing. Each version has these minimum capabilities:

- Full set of editor commands including saving search.
- Compact source lists, no line numbers. Free format text, no tabbing required.
- Read/Write named source files to cassette. Files may be merged.
- Superfast standard Z80 program assembler runs at four thousand lines per minute.
- Decimal, hex and word parameters.
- Six math operations: addition, subtraction, multiply, divide, ASCII, OR.
- Full range of pseudo ops for bytes, words, strings etc.
- Symbolic bit flags.
- Fully formatted LISA and Symbol Set to video or printer.
- Paging for almost local printers.
- Object to cassette or memory.

SHARP MODEM £19.50 + VAT
• £4.75k, runs under SP1002 or 1200k.
• Full object debugger included. Type in source, assemble then execute immediately.
• Debugger commands to set breakpoints, display Z80 registers, modify memory, etc.
• Read/Write/Verify source as object cassette.
• Jorden have produced heavily commented listing of SP1002 and SP1003 BASIC - please for details.

TR: 80 Level II £12.00 + VAT
• £4.5k, runs under LII at 1000k.
• Same facilities as Sharp version, requires EDITASM and TRUG.
• Cassette files in SYSTEM format.

NASCOM I £12.00 + VAT
• £2.5k, runs under NASCOM I, M800 or 050M.
• Cassette files in DUMP format.

STANDARD £12.00 + VAT
• £3.5k at 100k.
• Supplied on 300 baud level two cassette.
• Basic editor only monitor that recognises a carriage return.
• Cassette files in absolute level two format.

NEWBEAR COMPUTING STORE LTD.
40 Barbicanway Street,
Newbury, Berkshire.
RG14 5LL

ORDERS AND INFORMATION ON 0635 30905

1234

INTEGER PASCAL FOR A NASCOM

by PAUL WOOD
Supplied by
Dectron Micro Centre
Sheffield

Various BASICs are available for running on Nascoms, some of these are of the 'Tiny' variety. Integer Pascal is a 'Tiny' variety of Pascal which will fill the gap until you have CP/M on your Nascom and can then use one of the various 'Wirth Standard' Pascals.

To run Integer Pascal you will need at least 16k of RAM available. Two versions of Pascal can be supplied. One loads at 1000H and the other loads at 4000H. Both Nascom 1 and 2 tape formats can be supplied. Only NASSYS monitors are supported.

Enough of the advertising — now some technical details:-

Memory Allocation.

The object code for the package is 84K long and comprises a command processor, an editor, a two pass compiler and a series of runtime routines.

The address of the source area is set up at 'cold start' time.

1000H	INTEGER PASCAL
3000H	
3200H	
5000H	SOURCE AREA
7000H	
9000H	

After a 'cold start' a typical memory map would look like this.

Once a program has been created in the source area using the editor a 'syntax check' compilation may run.

1000H	INTEGER PASCAL
3000H	
3200H	
3800H	WORK AREA
5000H	PCODE
7000H	SOURCE AREA
9000H	

A 'syntax check' compilation runs pass 1 of the compiler which uses a work area immediately following the package and creates Pcode starting at 3800H. Upon successful compilation the highest address used for Pcode is displayed upon the T.V. screen.

When the extent of the Pcode area is known a 'full compilation may be run.

1000H	INTEGER PASCAL
3000H	
5000H	
7000H	PCODE
	OBJECT CODE
9000H	SOURCE AREA

A 'full' compilation comprises of both pass 1 and pass 2. Pass 2 takes the Pcode output by-pass 1 and translates it into Z80 object code. It is up to the user to provide the start address of the object code. The object code may not overlap the Pcode, neither may it be less than 4000H (this will be explained later).

Once a full compilation has been successfully compiled the program may be executed, either from the monitor with the standard command, or by a special Integer Pascal command.

1000H	INTEGER PASCAL
3000H	
3200H	
4000H	DYNAMIC WORK AREA
5000H	OBJECT CODE
7000H	
9000H	

During the execution of the object code various run time routines are called in the Integer Pascal package. Also memory is used in the dynamic work area, such things as variables, intermediate results, return addresses and more, are stored dynamically here. The dynamics work area starts at 3200H and extends as far as necessary, independent upon the number of variables, size of arrays, number of recursive calls etc. When a programme is executed the dynamic work area is zeroed for a length of 0E00H, therefore object code should not start below 4000H.

Due to the 'two pass' nature of the compiler it can be seen that the source program and object program are not handled together. This fact can be used to advantage on Nascoms with only 16K of RAM, or with very large programs. After creating the source program to the desired requirements save it on tape. Then run a full compilation over-writing the source area with object code. The object code may then be executed.

Control of the Package.

The package is controlled by entering commands. There are 13 commands allowing you to display, edit, compile and execute programs. See the Table of Commands for more details.

TABLE OF COMMANDS

A	DATA	Add line before current line
NNNN	DATA	Add line NNNN
B		Move current line pointer to beginning of source
C		Compile, Syntax check only
CNNNN		Compile, produce object code starting at address NNNN
D		Delete current line
DNNNN		Delete line NNNN
G		Go, execute program
L		List current line
LNNNN		List line NNNN
N		Renumber lines (increment 0010)
NNNNN		Renumber lines (increment NNNN)
P		Print source via VART
R	DATA	Replace current line
RNNNN	DATA	Replace line NNNN
S		Save source on tape
T		Terminate Package, return to NASSYS
Z		Clear Source Area
space		Display current line, next line becomes current line.

Remarks — Commands may be entered from anywhere on the screen. When lines are displayed they are in a suitable format for re-entry.

— A repeat keyboard facility has been implemented to allow easy movement of the cursor around the screen.

— NNNN is a hexadecimal number of up to four characters, DATA is the source code in question.

Integer Pascal Syntax

As is usual with descriptions of Pascal, refer to the syntax diagrams for an exact definition of valid syntax.

PEEKING and POKEing

Basic has its PEEKs and POKEs. Integer Pascal has an equivalent and more flexible method. Imagine the entire memory space as an array of 65536 elements. The array is called MEM, and can be used just like any other array.

MEM %OC29 : =%50; This code 'pokes' the cursor
MEM %OC2A : = %0A; address into the monitor work-
WRITE ('Display Text'); space. This results in 'Display
Text' being displayed on line
11 starting in column 7.

Because of the way Integer Pascal stores decimal values, the following table shows how decimal elements are to be defined —

MEM [0]	=MEM [%0000] = byte 0
MEM [32767]	=MEM [%7FFF] = byte 32767
MEM [32767]	=MEM [%8000] = byte 32768
MEM [-1]	=MEM [%FFFF] = byte 65535

Input and Output

Normally input and output will transfer decimal values e.g. READ (A) will request input of a decimal integer. Backspace allows you to correct input errors. Escape allows you to restart input and, as usual, New line causes the input field to be processed, the resulting value being placed in variable A.

To request ASCII input and output use a suffix of 'A'. To request hexadecimal input and output use a suffix of % e.g.

A = 65;
WRITE (A, ' ', A, ' ', A, '%')

will result in

65 A 0041
being displayed

CONST FF=%OC;
CR=%OD;
BEGIN
WRITE (FF, 'Heading', CR, 'CR', 'Another line')
END

will result in the screen being cleared and the following being displayed:-

```

1 |
2 | Heading
3 |
4 | Another line

```

Constants

ASCII constants are surrounded by single quotes. A single quote may be included in an ASCII string by the usual trick of the double quote, e.g. 'THAT' 'S MINE'.

Hexadecimal constants are indicated by a % prefix, e.g. %F800 and must be in the range %0000 to %FFFF.

Decimal constants must be in the range -32767 to +32767.

Arrays

Single dimension arrays are supported. The only limitation is the amount of memory available.

Final Remarks

Integer Pascal does have some restrictions. The main one is obvious from the name of the package — it does not support floating point arithmetic. In other respects it is a surprisingly full implementation of Pascal — it is recursive, assigns memory dynamically as required, supports procedures and functions, etc. etc. Please study the sample

program. Not only is it an excellent sort algorithm, but it also shows many features of Integer Pascal.

Sample Program

```
CONST NL=%OD;
VAR X: ARRAY [100] OF INTEGER;
    Y, Z: INTEGER;
PROCEDURE SORT (N);
PROCEDURE SRT (LEFT, RIGHT);
VAR A, B, C, D: INTEGER;
BEGIN
  A:=LEFT; B:=RIGHT;
  C:=(LEFT+RIGHT) DIV 2;
  REPEAT
    WHILE X[A] < C DO A:=A+1;
    WHILE C < X[B] DO B:=B-1;
    IF A < B THEN
      BEGIN
        D:=X[A]; X[A]:=X[B]; X[B]:=D;
        A:=A+1; B:=B-1;
      END
    UNTIL A > B;
    IF LEFT < B THEN SRT (LEFT, B);
    IF A < RIGHT THEN SRT (A, RIGHT)
  END;
END;
```

```
BEGIN
  SRT (1,N)
END;
PROCEDURE NEWLINE (X);
BEGIN
  IF X MOD 8 = 0
    THEN WRITE (NL)
    ELSE WRITE ( ' ')
END;
BEGIN
  WRITE ('Enter values:', NL);
  Y:=0;
  REPEAT
    Y:=Y+1;
    READ (X [Y]); NEWLINE (Y)
  UNTIL X [Y] = 0;
  SORT (Y-1);
  WRITE (NL), 'sorted values are:', NL)
  FOR Z:=1 TO Y-1 DO
    BEGIN
      WRITE (X[Z]); NEWLINE (Z)
    END
  END.
```

The program reads values into array X, sorts the array and displays the values in ascending sequence. The method used is 'Quicksort'. Note the recursive calls of procedure SRT.

New and exciting Applesoft programs

NEW! the correspondent

by R. Wagner

THE CORRESPONDENT is sure to be one of the most versatile programs in your library! It can be used as:

A Text Processor: Upper/lower case, 1-80 cols. (4-way scrolling). Text move/copy/insert/delete, tabbing, justify text, auto-centering and more!

A Database (with or without printer!) Extremely fast find routine and easy editing make it a natural for free-form data files. Create and fill out forms, access phone lists or index your magazines.

A Programming Utility: (printer or not). Examine, edit, transfer random or sequential text files. Create versatile exec. files. Even put bi-directional scrolling in your own programs!

Apple disk £29.95 + VAT

Roger's Easel

by R. Wagner

At last a program which allows you to draw colour pictures in lo-res graphics, and then permanently link them to your own Integer or Applesoft programs. Linked pictures can be displayed on either text/graphics page. (Integer basic).

Apple disk £14.95 + VAT

Apple-Doc

By Roger Wagner

An Aid to the Development and Documentation of Applesoft Programs

This 3 program set is a must to anyone writing or using programs in Applesoft! It not only provides valuable info. on each of your programs, but allows you to change any element throughout the listing almost as easily as you would change a single line!

With Apple-Doc you can produce a list of every variable in your program and the lines each is used on, each line called by a GOTO, GOSUB, etc., in fact, every occurrence of almost anything! You can re-name variables, change constants and referenced line numbers or do local or global replacement editing on your listing.

Apple-Doc is a must for the serious Applesoft programmer.

Diskette complete with full documentation £24.95 + VAT

tridee

Procedures include Ortho, Perspec, Rotate, View, Move to 3, View from.

An exciting new addition to your Pascal library — enables you to create 3D graphics, viewable from any angle and distance. As easy to use as Turtlegraphics.

Complete with comprehensive instructions £49.95 + VAT

Apple World

is here. The fast 3D graphics package that runs on your Apple II plus. Zoom, pan, tilt and scale your own designs on the Apple screen, at only £24.95 + VAT

Plus a complete range of "off the shelf" programs for finance, commercial, scientific and education. Keep yourself up to date, send for our "Fact Sheets" giving full program details.

Now available Apple FORTRAN. Dos 3.3, Apple Plot

LEICESTER
computer centre limited

67 Regent Road, Leicester LE1 6YF. Tel: 0533 556268

apple computer
Sales and Service



Programming Practices and Technics

STRUCTURED AND NOT SO STRUCTURED PROGRAMMING

Martin D. Beer,
Computer Laboratory
University of Liverpool.
P.O. Box 147
Liverpool,
L69 3BX



STRUCTURED Programming is a common theme which runs through much of the microcomputing press. To the novice programmer, intent to get his first new programs to work, it can all be very confusing. Not only does he have to learn a new language in which to write his programs, BASIC, PASCAL and the like, but there are also all the concepts of how a program is built up in the form of sequences and loops and conditions, which, at first, does not appear to help him in his primary task of writing a program.

In many ways the skills needed to write a computer program are the same as those needed to write English. Learning the language is much the same. To write in English requires a comparatively small vocabulary of the most frequently used words, and a much larger optional one which covers the things that you wish to write about. The rules of sentence construction, and punctuation, must be understood, and applied with great precision, otherwise the meaning of what you write may well be misunderstood by the reader. Computer Languages are like this also. They consist of the basic construction and punctuation rules and a very basic vocabulary of commonly needed operations and other useful items. The difference is that the optional vocabulary is not selected from a standard dictionary, since this would be too restrictive, but the programmer must write his own in the form of declarations at the start of the program (in some languages, such as FORTRAN and BASIC it is not necessary to explicitly declare variables, so long as certain rules are applied. In effect, these variables are implied implicitly at the start of the program).

So far so good, but if we apply the basic rules of English mindlessly we can write nonsense, even though each sentence is perfectly intelligible. This can be avoided by drawing out a plan of what you wish to say, and steadily refining that plan, as you write so as to obtain a clear text which conveys to reader exactly what you wish to say. Headings and paragraphs are used to split the text into manageable chunks which can be read, and digested before moving on to the next section. Each sentence adds to the information which you wish to convey to the reader.

With a program there are two readers, the computer, which must fully comprehend what you are telling it to do, and other programmers, who will read the text of the programs to find out what you were trying to tell the com-

puter to do. These are somewhat different aims, and require various compromises to be made. It is impossible for most people to fully comprehend more than a few paragraphs at any one time. This article, for instance, although when finished will be between 1000 and 2000 words long, has written in comparatively short sections of only 100-200 words over a period of several weeks. You will probably read it as a series of sections which are slightly longer than this. The computer is not particularly worried about the length of the program section it is required to execute, but remembers every connection in that section, whereas the human reader is much more discriminating, in that he does not usually make connections between items widely separated in the text. It is therefore essential to split the program into sections, similar to the paragraphs of the English text, which are totally independent of the rest of the code. This is done by splitting the program into a series of procedures which can be written separately, and perform a single, clearly defined, part of the program.

different items which you define in the programme so long as the name is used consistently throughout the program. It does help the human reader, however, if all names are chosen in such a way that they clearly indicate what is your intention to represent.

So the line of code

Force := mass * acceleration
is a much more meaningful than
f := m * a

It is also important to choose names which can be clearly differentiated. If the names are only different by the last letter, or two, it may well prove difficult to read (and many compilers, particularly those which actually run on microcomputer systems only recognize the first few letters of each name, so as to save space. So if, as in the case of the UCSD PASCAL compiler only the first eight characters are recognized the following would not be differentiated:-

acceleration 1
acceleration 2
acceleration 3

or:

initial time
initial test
initial temp

This is sometimes not all that obvious, since the human reader recognises the differences, no matter how long the word is.)

A related point is that it is inadvisable to use variable, and other names, to represent more than one thing in any program section. It may appear to save memory if a variable is used to store different data at different points in the program. This can cause curious side effects, if the program is modified, since the variable may well not contain what was expected at that point in the program. If each separate data item is stored in its own variable location this problem will not arise.

Structured Programming is the generic term which covers a wide variety of techniques aimed at making programs much more readable, and easier to modify, should be necessary as user requirements change. It has long been recognised that programs can be built up from three basic elements:-

- a) sequences of statements
- b) statements which are executed only if some specified condition is met
- c) groups of statements which are executed repeatedly so long as some condition is met.

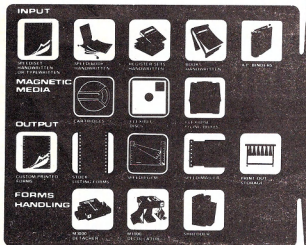
This is the underlying structure of the program, which should be brought out in its text.

The program is split into two major sections, which are analogous to the chapters of a book, which are then subdivided into subsections, and finally paragraphs. Each of these divisions is represented within the program text as a procedure, or functions. Each procedure has a definite specification, which is all that the programmer needs to know about when writing the higher levels.

This allows the programmer to concentrate on a convenient slice of the program when designing and writing the code, thus avoiding much of the confusion which can be felt by new programmers when writing their first program.

As with any form of communicating, style of presentation is important in programming, but whereas in writing and conversation we are trying to express our ideas to another human being, when writing a computer program we are not only communicating ideas, but are also giving very specific instructions to a machine to perform some special task. It will not necessarily interpret the instructions given to it in exactly the same way a person would, and therefore a style should be adopted which will make the task as simple as possible for all concerned.

MSBC TOTAL PACKAGE



THE TOTAL MOORE PARAGON FOR YOUR SMALL COMPUTER

**ALL YOUR
PRODUCT
NEEDS FROM
ONE SUPPLIER...**

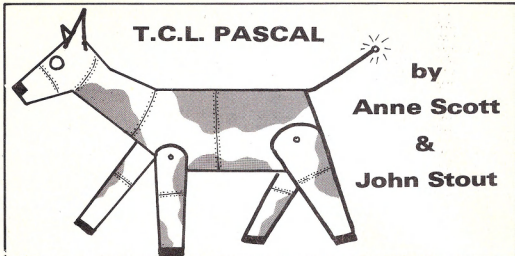
TIME SAVING...

COST SAVING

FOR ALL YOUR
COMPUTER STATIONARY
REQUIREMENTS

FOR FURTHER DETAILS:
MOORE PARAGON LTD.,
PIONEER HOUSE,
16 CROSBY RD NORTH
L22

Tel: 051 928 4305



Description.

What you get.

For £120 the purchaser of the TCL Pascal system gets one mini-floppy disk for use in a Commodore disk drive, a manual, a security ROM and a licensing agreement. Plug the security ROM into the \$A000 ROM socket, load the mini-floppy into drive 0 and type:

```
LOAD " " , 8
```

The drive will be initialised (close the drive door during the 'clunking' sound) and the first file on the disk, aptly named PASCAL, will be loaded. Type:

```
RUN
```

and the first of the two compilers supplied with the system will be loaded into the PET, together with an editor for preparing programs. Programming in Pascal can now begin.

Editor Commands.

To enter a program into the machine's memory it is typed in using much the same procedure as with BASIC. Program lines are preceded by a line number and are inserted into the existing program in the correct position with relation to other lines already in memory. Entering just a line number and the RETURN deletes that line from the memory.

NEW and LIST work just as in BASIC, but only BASIC command per line is allowed, so that:

```
OPEN 4, 4: CMD 4: LIST
```

doesn't work. Split the commands up on to separate lines and the problem is solved. A listing is more conveniently generated by using the P command once any compilation errors have been removed.

PUT and GET are like the BASIC SAVE and LOAD commands, but need no device number or quotes (" ")

around the filename. The filename can include a drive specifier, e.g. GET 0: ASSEMBLER, PUT 1: SCROLLER, and in fact using a drive specifier was found to be almost obligatory for PUT, to avoid generating files with names like @DIRECTORY.

The editor provides several facilities similar to those in the BASIC Programmers' Toolkit and the editor provided with the ASSEMBLER package from Commodore: DELETE allows the block deletion of lines in the program, using a syntax like that of LIST, FIND finds all occurrences of a specified string, the string being delimited by the first non-space character after FIND, e.g.

```
FIND /WRITELN/
```

CHANGE not only finds the occurrences of a string but replaces them with another specified string. The two strings are delimited in the same manner as before, e.g.

```
CHANGE @ DIV @/@
```

changes all instances of DIV (preceded and succeeded with a space) to /, i.e. integer to real division. Both FIND and CHANGE can be followed by a line number or range of line numbers (specified as with LIST) in order to restrict their operation. For example:

```
FIND $WRITE$, 150-200
```

```
CHANGE +WRITELN+WRITE+, 100-
```

AUTO specifies the increment to be used between line numbers, which are generated automatically, as in TOOLKIT, except that the first line number must be entered by the user. The default value is 10, and AUTO without a number switches off the facility.

NUMBER rennumbers the lines of programming and has the syntax:

```
NUMBER (old first line), (new first line), (step).
```

It will renumber starting with the (old first line) and making it (new first line) then increasing by (step) for the rest of the lines in the program. The (new first line) must be

greater than or equal to the (old first line) which is a nuisance especially with errors as we shall see below. NUMBER has no effect on GOTO statements in Pascal since these statements refer to declared labels which are independent of line numbers. The line numbers do not form part of a Pascal program — they are simply for convenience of editing.

Other commands are:

UPPER and LOWER which, as the names imply, switch between upper and lower case, BASIC, which returns to the normal BASIC system (should you ever want to), when Pascal can be reloaded with:

PASCAL (load and run using the DOS SUPPORT up arrow command),

BREAK gets the user into the normal PET resident monitor, HEX and DECIMAL provide for conversion for base 16 to base 10 and vice versa.

All DOS SUPPORT commands are provided, together with a repeat function on all keys, which is especially useful for editing. BASIC commands PRINT, PRINT #, OPEN, CLOSE, CMD, POKE, SYS, FOR and LET are also supported.

Compiler Commands.

As mentioned earlier the package supplies two compilers, the first of which, the resident compiler, is loaded in with the LOAD " * ", 8 command. This compiler may be recalled at any time by using the RESIDENT command (although a large program may have to be deleted or saved to disk). As its name implies it remains in memory together with the pascal program and works in much the same way as the PET's BASIC interpreter. Type RUN (or R) with a program in the memory and the program is compiled to p-code (see below). If no errors are detected the resulting program is executed, and may be repeatedly executed by typing RUN (or R) as long as no alterations to the program are made. The disk is not used unless an error is detected, when the appropriate message is read from the file PASCAL ERROR MSG on the disk. L and P compile the program without running but provide a listing to the screen or printer respectively.

The disk command paves the way for the disk based compiler by deleting the resident compiler (it also allows more space for programs during editing). COMP then compiles a program on the disk, specified with a drive number if need be, e.g.

COMP 1: COUNTER

An object file, containing the p-code produced by the compiler, is then created on drive 0 with the .OBJ extension, in the example above COUNTER.OBJ. This file may then be executed using the EX command. COMP may also specify one or more of the following options:

1 (object file on drive 1)

C (no range Checking or line numbers in the .OBJ file, giving a slightly faster and more compact program)

L (produce a Listing on the screen)

N (No object code produced, e.g. just check for errors)

P (produce a listing on the Printer),
e.g. COMP COUNTER, 1, C or COMP PROG 2, N, P.

LINK may be used to link together separately compiled functions or procedures with a main program. Thus a library of useful 'routines' may be built up, which can cut down the time for program development.

The # (hash) sign facility can be used in a program as the first character on a line when followed by a filename, to include the contents of the file named in the program being compiled. Thus a common declarations segment can be included in several files, and only one file need be edited to incorporate any changes in all the files.

LOCATE may be used to create a file from a .OBJ file which can be LOAded and RUN from BASIC. The only BASIC line available for listing will be a SYS to the start of the p-code interpreter. According to the manual programs can be made in this way to 'run on a 16K PET, since the runtime interpreter size is only 10K'. Thus we might see the appearance of programs written in Pascal for non-Pascal systems, although the run-time interpreter will not work without the security ROM, on sale from TCL for £20. This seems a rather unfortunate decision, since it might put the price of programs up unnecessarily.

The cycle of operations involved in producing a Pascal program using the disk based compiler is thus:

- 1) design program
- 2) code it into Pascal
- 3) insert into memory using the editor
- 4) save to disk using PUT
- 5) compile using COMP
- 6) if there are any errors load program using GET and edit again. Then return to step 4
- 7) execute program using EX when no compilation errors
- 8) if there are any run-time errors, or the program doesn't work, use GET again, correct errors and return to 4
- 9) produce a stand alone program using LOCATE if needed.

One of the major attractions of Pascal, if used properly, is that the number of cycles though the above stages should be fewer than with other languages, although the time spent in stages 1 and 2 may be longer.

Evaluation

Problems.

1. There is no VERIFY command built into the editor, so that the first time you know a program has not been written correctly to disk is when you try to read it back — too late. Moral — PUT at least two copies, preferably on different disks, of any long programs which are not just minor updates.

2. The resident compiler is very useful for developing programs since it takes time to GET and PUT programs and to read in the disk based compiler.

However it will not support very large programs or ones accessing disk files in the Standard Pascal way, although the IEEE bus can still be accessed easily. The procedure DISPOSE is not supported in resident mode, which since space is tight anyway can cause problems with pointer variables.

3. Because TCL Pascal uses the disk so intensively, any unresolved problems with the disk operating system may have far reaching effects. We have frequently had corrupted disk files and system hang-ups which appear to be caused by DOS bugs.

Corrupted files are probably caused by the fact that @ is automatically prefixed the name of each file created, so as to overwrite any existing file of the same name. Use of this overwriting facility is known to cause problems in some circumstances.

So far we have evolved the following rules for creating files:

- always include drive number in filename (otherwise you can find you have a file called @FILE1 for example),
- always scratch any existing object file of the same name before COMP or LINK. Use the .N option for COMP if possible,
- never execute REWRITE on any particular file more than once in a program run,
- make sure any old versions of external files are scratched before running a program which writes to external files (internal files are automatically scratched at the end of the program block within which they are defined).

The best solution seems to be to use a different name each time a new version of the file is produced, e.g. MYPROG 1, MYPROG 2, MYPROG 3 etc. This can be done within programs since filenames may be assigned dynamically:

```
var
  FILENAME : packed array [1..12] of char;
  (* i.e. a string 12 characters in length *)
  F : text;          (* i.e. a file of char *)
  EXIT : boolean;

begin
  FILENAME := '1: DATAFILE0'; (*must end with a
  space *)

repeat
  .
  .
  FILENAME [11] := succ (FILENAME [11]);
  rewrite (F, FILENAME);

until (FILENAME [11] = '9') or (EXIT);
```

end. (* upper case characters represent user defined identifiers, lower case characters system defined, standard, identifiers, and underlined lower case characters represent symbols of the Pascal language, e.g. begin, as do symbols such as =, := *)

This allows up to 10 versions of DATAFILE to be created. By initialising the name to DATAFILE0 then up to 26 versions could be produced. Note that uppercase letters only are used in filenames.

Another problem with disk files occurs because there is

no explicit way in Pascal to close a file. This is done automatically at the end of the block within which the file is defined, but sometimes a disk error can cause the system to hang up. Disconnecting the disk may return you to the editor or you may have to switch the PET off. In either case any disk files being written to may not have been properly closed and can cause havoc later. Reconnect the disk unit and type:

```
OPEN 15, 8, 15
PRINT #15, "I"
CLOSE 15
```

Since input/output errors can be trapped within a program by use of the procedure iotrap and function ioerror offending files can be closed by means of a dummy reset statement, e.g.

```
reset (F, 0, 0);
```

which resets the file to reference the keyboard (IEEE 'device' 0) before returning to the main program flow.

4. Input of numbers and strings (defined as packed array [1..LENGTH] of char) can be accomplished in much the same way as in BASIC, but using READ not INPUT. Unfortunately a number of programs refused to work when requesting single character input from the keyboard. For example the program to reverse words on page 37 of the manual does not work until one 'fiddles' it to reverse the entire sentence at one go, by changing until input? = ' '; to until eoln; The algebraic to reverse Polish program in Jensen and Wirth's Pascal User's Manual does not work either, at least with keyboard input. Apple Pascal has a special type of file, of type interactive, to cope with some of the problems associated with interactive programming and this is possibly something TCL could consider for later versions.

5. The diagnostics are in general good, since most errors will be caught at compile time (this is one of the reasons it is easier to write correct programs in Pascal). However, as with all block structured languages the error may be a few characters out, and the compiler points this out by declaring errors as being 'near X', or whatever the last symbol scanned was. The line numbers given in error messages can be confusing, since when a program is saved to disk using PUT the line numbers used by the editor are not saved as well. GETTING a program results in one starting at line 1000 in steps of 10. However the disk compiler produces error messages referring to lines starting at line 1 in steps of 1, which can be confusing. This means that it is almost essential to use one of the listing options when compiling, usually the P option being the better of the two if you have a printer.

6. Since LINK operates on compiled routines it cannot check for matching of number and types of parameters in externally declared routines. Errors of Matching of parameters between the actual calls and the declaration in the calling section will be made, since the procedure declaration must include parameters, but the checking cannot be done between the calls and the external procedure called. This problem is not unique to TCL Pascal, but is included here for completeness. A similar problem occurs with passing parameters to procedures or functions which are themselves functions or parameters, in which case the correspondence of parameters is not checked either.

Advantages.

Despite the long list of problems, much of which is devoted to the problems associated with files, the overall impression gained from the package is favourable. TCL Pascal is the first 'standard' high level language to make its appearance for the PET. As can be seen from the benchmark timings elsewhere it does not provide a spectacular increase in speed for all applications, but for some of the non-computational problems encountered the gains can be considerable. For example, if we have two arrays of real numbers, say 30 by 30, the BASIC programming to transfer one to the other uses two nested loops and two extra variables. The corresponding Pascal statement simply assigns one variable to the other, e.g. `A := B`. Since the sizes of the arrays will be fixed at compile time the p-code for this statement will probably be only 4 instructions, 10 bytes long, and will result in the transfer being done almost at full machine code speed. BASIC takes 10 seconds while Pascal takes < 1 second, and is transferring 900 more bytes than BASIC, since reals in Pascal take 6 bytes not 5 as in BASIC.

The system is very easy to use and the resident compiler helps a lot when getting short sections to work. It can lead to programming in much the same way as programming in BASIC, repeating the edit-run-edit cycle rather than the think-edit-run cycle. The presence of line numbers might provide a useful stepping stone for those people working their way up from BASIC. The software proved robust, with the exceptions concerning files mentioned earlier. Since these seem most likely to come from Commodore's software rather than TCL's it would be interesting to see a version adapted to the new BASIC 4.00 and DOS 2.1. We found the best way of working was to leave the system disk, or rather a copy of it, in drive zero and work on a work disk in drive 1.

The only execution error reporting is a message and a line number. No dump or trace facility is provided, but then again neither do many other versions of Pascal.

Extensions.

There are many extensions to Standard Pascal, which may or may not be a good thing. Most are obviously derived from BASIC, e.g. peek and poke, page and getkey. The first two should be obvious, page clears the screen, while getkey is a function returning a character, like `GET A$` in BASIC. The BASIC line `100 GET A$: IF A$ = " " THEN 100` can be replaced by `repeat KEY := getkey until KEY < chr(0)`; if the key pressed must be remembered, or while `getkey < chr(0)` do: if it need not be remembered. The procedure VDU allows the poking of characters to the screen, but to a particular row and column and with automatic conversion to screen code if need be. A number of facilities for working base 16 are provided, as is a facility for treating integers as logical quantities and applying the logical operations of and, or, not and xor to them. Shifting left and right is also provided. The stop key may be disabled by a call to the procedure BREAKS, a random number generator function is supplied and the PET's internal clock can be set and tested, but not to the accuracy of a sixtieth of a second available in BASIC.

Devices on the IEEE bus can be controlled easily using

versions of reset and rewrite, and the printer can be supplied with automatic lower/upper case conversion facilities.

As has been mentioned the type **packed array** [1..LENGTH] of char; can be treated as a string, but the treatment does not include the sort of facilities available under BASIC for concatenation, substringing etc. Some facilities can be mimicked using the variant record facility, but all suffer from Pascal's insistence on knowing the size of everything at compile time. A string facility such as that of Apple Pascal (UCSD Pascal more properly) would be useful, but would no doubt become prohibitive in terms of size.

One of the most interesting extensions is that provided by the origin procedure which allows the setting of a pointer variable to any value. The example given in the book supplied (which could do with an index and a list of recommended reading) is of pointing to the screen and accessing it through a packed array [1..1000] of char; A more useful way would be to reference it through an array [1..25] of packed array [1..40] of char; so that individual rows can be treated as separate entities. The scroller program illustrates this. Another example could arise by defining types and variables as below:

```
type
  VIAREGS = (PORTB, PORTAH, DDRB, DDRA,
             TICL, TICH, TILL, TIHL, TC2L,
             TC2H, SR, ACR, PCR, IFR, IER,
             PORTANH);
  VIACHIP = packed array [PORTB, PORTANH] of
    0..255;
var
  VIA : ^VIACHIP;
```

Then a call to origin of:

```
origin (VIA, $E850);
```

allows accessing the registers of the PET's VIA as follows:

```
VIA↑[DDRA] := 15; (* bits 0..3 output, rest input*)
VIA↑[SR]    := 15; (* 50% duty cycle in shift register*)
and so on.
```

While writing the assembler routine to find the set limits, it was found that the representation of the set passed as an argument to the routine was not as detailed in the manual, which states that the 16 bytes used are split into pairs, thus forming 16 bit 'integers' and that each of the 8 pairs thus formed is stored in normal 6502 format, i.e. with lower order byte first (lowest in memory), and with the highest order 16 bits, i.e. for elements with ordinal values 112 to 127, first, i.e. lowest in memory. In actual fact the set is represented as 16 consecutive bytes, the lowest in memory encoding elements with ordinal values 0 to 7 (bit 0 encodes element with ordinal value 0), the next highest encoding elements with ordinal values 8 to 15 and so on.

Timing Benchmarks (TCL Pascal against CBM BASIC)

Benchmarks BM1 to BM8 (published in October's Personal Computer World) were run on a standard PET in BASIC versions and on a PET running the resident TCL Pascal Compiler for the BASIC versions the timing was done by setting `TIS = "000000"` (which resets TI to 0), executing

the benchmark, then printing out the value of TI/60 which gives the time taken in seconds.

The Pascal versions set the time using the procedure setting (0,0,0) to zero, ran the benchmarks and the printed minutes * 60 %seconds. Depending on whether the internal clock procedures round up or down this could be up to 1 second out in either direction.

For the BASIC times two figures are given, the first time for the benchmarks entered as written (one statement per line, spaces between keywords etc) and the second (in parentheses) for the benchmarks compacted as much as possible (as many statements as possible, no unnecessary spaces). For those benchmarks involving division the Pascal times give four values. These times come from the four possibilities of data type for the left hand side of the assignment statement (integer or real) crossed with the two possible division operators possible (div or /). div acts on two integer operands and discards the remainder, e.g. (25 div 7) = 3. / acts on either two integer variables, two real variables or one of each type. In all cases the trunc function was applied if necessary (to convert from type real to type integer) and the variable K (used to count loops) was always integer although in benchmarks 2 to 8 it could have been real (since a for statement was not used). This approach to the benchmarks is valid (I think) since with Pascal one can tailor the data types according to the application. Certainly the figures show that the / operator consumes a lot of the time in the benchmarks, so that if the data typing of the operands allows using div then substantial gains can be obtained. Similarly the removal of spaces in the BASIC benchmarks shows that this technique gives little benefit in terms of speed.

Benchmark	BM1	BM2	BM3	BM4	BM5	BM6	BM7	BM8
BASIC	1.4 (1.3)	9.6 (9.1)	18.2 (17.5)	21.9 (21.2)	23.6 (22.7)	34.1 (32.9)	52.5 (51.1)	11.6 (11.5)
Pascal	1	1	10	9	10	16	27	5
			4	6	7	13	23	5
			10	9	10	16	27	
			5	6	7	13	23	
							21	
							18	
							25	
							22	

The entries for BM7 occur because the array A used was declared as **array** (1..5) of real for the first four entries, and as **array** (1..5) of integer for the second four entries. The two entries under BM8 come about by declaring a variable A: = **sqr** (K) as either real or integer.

Compilation times have not been included, since they are not really relevant to the question of how fast the program executes. Once the program has been compiled the object code can be executed immediately without going through the compilation process again.

Last minute points.

Chaining of programs is possible, using the chain (FILENAME) command, either to another Pascal program, in which case variables are preserved only if both programs have identical declarations at the global level, or to BASIC programs.

Sets are limited to having only up to 128 members, i.e. **Ord**(SETMEMBER) < 127.

Linkage to assembly language programs, functions or procedures is possible, although the information given and the method of implementation give the impression of having been tacked on at the last minute. As in any external procedure/function in the system a duplicate procedure/function header is inserted in the main program and the body replaced by the keyboard extern. However, for assembly language programs the absolute address of the resulting machine code is added, e.g. **procedure SETLIMITS** (ASSET : **NUMSET**; var **SETLIMITS**: **LIMITARRAY**); **extern** \$7432; Since the only information about the procedure is its start address the compiler cannot load the machine code, and the code must be loaded 'by hand'. The easiest way we found was to **LOCATE** the main program, load it as normal, then load the machine code and alter the end-of-memory pointer to protect the machine code (at \$34, \$35). The two examples below, in hex dumps, each operate on sets, the first one returning a count of the number of elements in the set, while the second (a procedure whereas the first is a function), uses a **VARIABLe** parameter to return two results, the ord of the 'first' element in the set, and the ord of the 'last' element in the set. Thus the values 65 and 85 would be returned with a call using the set ['A', 'E', 'I', 'O', 'U']. The function has heading:

```
function SETCOUNT (ASSET : set of ***): integer;
extern $7401;
```

```
while the procedure has heading:
procedure LIMITSOF (ASET: set of ***; VAR SET-
LIMITS: limitsarray);
extern $7432;
```

```
where the following typedeclarations apply:
type LIMITS = (MINIMUM, MAXIMUM);
LIMITSARRAY = array [ MINIMUM .. MAXIMUM] of
integer;
```

In both cases *** represents some basetype for a set. Note that a separate declaration of both **SETCOUNT** and **LIMITSOF** must be made for each type of set to be examined with a different name in all cases, but with the same procedure/ function body in each case, since the machine code circumvents Pascal's strict typing rules. The procedure **LIMITSOF** will be discussed in the next issue in more detail, together with a proof of its correctness. If **ASET** is the null set, both elements of the away **SETLIMITS** are set to 128.

Variables are not initialised to afe values (e.g. 0, 0, 0, false) when they are declared. Programs depending on them being so initialised (bad practise) will not work. The version number of the Pascal we used was version 1.5. We would like to thank Tansam Ltd for providing us with a review copy of TCL Pascal, and Keith Frewin for answering a number of our questions.

To sum up, a very good means of stepping across from BASIC to Pascal. In the resident compiler a lot of BASIC skills are still used which should reassure those new to compiled languages. Some rough edges, especially with respect to files and to a lesser extent with assembler linkage. However the need to program in assembler should be reduced a lot with the availability of the system.

```

1000 xprogram utilitel(input,output);
1010 (*written anne spott 11/9/82
1020 (*a routine which can be linked
1030 (*there should be no outer level
1040 (*declare xprocedure xclickkey as below
1050 xprocedure xclickkey
1060   xkeyset:=set of char;
1070   var key char;
1080   (*sets keypress in given set*)
1090   var i(integer);ch(char);
1100   begin
1110     for i:=1 to 10 do ch:=xkeyset[key];
1120     while not(ch in keyset)do
1130       begin
1140         if ord(ch)>0 then
1150           begin(*wrong key*)
1160             for i:=1 to 10 do ch:=xkeyset[key];
1170             writeLn('Wrong key - try again');
1180             and
1190             ch:=xkeyset[key];
1200             end(*while*);
1210             key:=ch;
1220             end(*clickkey*);
1230 xprocedure xktest;
1240   var key char;
1250   begin
1260     writeLn(''''S''' means '''stop'''');
1270     repeat
1280       writeLn('Press A C P or S. ');
1290       xclickkey(['a','c','p','s']/key);
1300       writeLn('You pressed ',key,' ');
1310       until key='s';
1320     end;
1330 begin(*main body*) xktest end.

```

with a main xprogram,
variable declarations,
with 'extern' for body. *)

Count elements
in a set
<(\$7400-\$742f).

Setrange
procedure
<(\$7430-\$74a3).

```

B*
PC IR0 SR AC XR YR SP
.. 0401 E62E 32 04 5E 00 F8
..
.. 7400 80 A0 00 8C 00 74 A2 03
.. 7408 B1 2A F0 09 0A 90 03 EE
.. 7410 00 74 C8 D0 F7 C8 C0 10
.. 7418 D0 EE 98 A9 00 91 2A 98
.. 7420 AD 00 74 91 2A 98 18 65
.. 7428 2A 85 2A 90 02 E6 2B 60
..
.. ?

```

```

B*
PC IR0 SR AC XR YR SP
.. 0401 E62E 32 04 5E 00 F8
..
.. 7430 00 7E A9 7F 8D 31 74 A9
.. 7438 00 8D 30 74 A8 B1 2A 38
.. 7440 E9 04 85 01 C8 B1 2A E9
.. 7448 00 85 02 C8 B1 2A D0 32
.. 7450 18 AD 30 74 69 08 8D 30
.. 7458 74 C8 C0 12 D0 EE 8D 31
.. 7460 74 A0 00 AD 30 74 91 01
.. 7468 98 C8 91 01 C8 C8 91 01
.. 7470 88 AD 31 74 91 01 A9 12
.. 7478 18 65 2A 85 2A 90 02 E6
.. 7480 2B 60 4A B0 05 EE 30 74
.. 7488 D0 F8 A0 11 B1 2A D0 0C
.. 7490 38 AD 31 74 E9 08 8D 31
.. 7498 74 88 D0 F0 0A B0 C2 CE
.. 74A0 31 74 D0 F0 00 AA AA AA

```

A USEFUL PASCAL PROGRAM!



#2 **Dr. Martin D. Beer**

PROGRAM QUADRATIC1

USES TRANSCENDENTAL;

VAR

A,B,C:REAL;
ROOT1,ROOT2:REAL;
SROOT:REAL;
CH:CHAR;

BEGIN

REPEAT

PAGE(OUTPUT);

Writeln;

Writeln('PROGRAM TO FIND THE ROOTS OF A QUADRATIC');

Writeln;

Writeln('EQUATION OF THE FORM');

Writeln;

Writeln('AX² + BX + C = 0');

Writeln;

Writeln;

Write('ENTER A --');

Readln(A);

Write('ENTER B --');

Readln(B);

Write('ENTER C --');

Readln(C);

Writeln;

SROOT:=SQRT(B)-4.0*AC;

IF SROOT=0.0 THEN

BEGIN

ROOT1:=-B/(2.0*A);

Writeln('THERE IS ONLY ONE REAL ROOT,');

Writeln;

Writeln('AND IT IS - ',ROOT1);

END

ELSE

IF SROOT<0.0 THEN

BEGIN

SROOT:=SQRT(SROOT);

ROOT1:=(-B+SROOT)/(2*A);

ROOT2:=(-B-SROOT)/(2*A);

Writeln('THERE ARE TWO REAL ROOTS,');

Writeln;

Writeln('AND THEY ARE - ',ROOT1, ' AND ',ROOT2);

END

ELSE

BEGIN

SROOT:=SQRT(ABS(SROOT));

ROOT1:=-B/(2*A);

ROOT2:=SROOT/(2*A);

Writeln('THERE ARE TWO IMAGINARY ROOTS,');

Writeln;

Writeln('AND THEY ARE - ',ROOT1, ' + OR - ',ROOT2);

END;

Writeln;

Write('ANOTHER PROBLEM ? (YES OR NO --)');

Readln(CH);

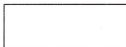
Writeln

UNTIL CH<>'Y';

END.

Free Classified Ads!

Fill this box
like this



370 professionally assembled,
tested and working, system
software with source code, offers
or possible FX Mk 14, buyer
collects 01-234 5678

Send your ad typed
or even hand written
(max approx. 20 words)
to Steph at
14 Castle Street
Liverpool L2 0TA
or phone
051-227 2535

Insertion at editorial discretion

Pets Corner



J. Stout

Department of Architecture
Liverpool Polytechnic
53 Victoria Street
Liverpool
L1 6EY

Three errors, two in the typesetting and one in the programming, crept into the hexadecimal dump of the relocater in the last issue. First the typesetting errors:

location S185B should be 30 not 80
location S199C should be 73 not 83

The programming error occurred in locations S18CC to S18E4, which should read:

```
38 A5 C1 E5 BD 85 B1 A5 C2 E5 BE 85 B2 A9 0C
20 02 18 90 05 A9 0E 20 02 18
not
A9 0C 20 02 18 90 05 A9 0E 20 02 18 38 A5 C1 E5
BD 85 B1 A5 C2 E5 BE 85 B2
```

The error would only have occurred with commands which specified a range of addresses within which addresses in the program were to be altered, and where the first address was not the same as the first address (old from) in the block to be relocated.

Apologies to all those who have had problems.

The promised review of TCL's Pascal for the PET appears elsewhere in this issue.

SETS

The concept of a set is central to mathematics (given the concept of a null set, that is a set containing nothing, one can define the natural numbers, and from them go on to deduce the rest of mathematics), but the set can have its uses in programming, as this section attempts to show. The programming language Pascal has as one of its fundamental data structures the set, and its use can lead to quite elegant programs and algorithms.

A set is a collection of objects, the order not mattering, the objects being chosen from a 'universe' of objects. Thus a universe for one particular application might

be the digits 0 to 9, and we would be able to define a set on that universe by saying for each element of the universe whether or not it was in the set. Thus for a set which contains only the even digits we would say for 0 yes, for 1 no, for 2 yes, 3 no, 4 yes, 5 no, 6 yes, 7 no, 8 yes, 9 no. For each element in the universe we are making a binary decision with the outcomes yes or no, so we could represent the set by a string of binary digits, the string being as long as the number of elements in the universe. Putting the 10 decisions above into a single binary number gives us:

1010101010, or in decimal 682,

although for reasons that should become obvious later we reverse the order and get:

0101010101 or in decimal 341

Given a number of sets defined on a given universe, represented by A, B, C etc, there are a number of questions that may be asked about these sets, their relationship with each other, and with other elements of the universe, represented by a, b, c etc. Using the binary representation (b.r.) developed above these questions become comparatively easy to answer.

For example:

- given two sets A and B, what is the intersection of those two sets, that is the set C which consists of elements which are in set A and in set B.
- given two sets A and B, what is the union of those two sets, that is the set C which consists of elements which are in A, or in B, or in both A and B.
- given an element of the universe e, and a set F, is the element e a member of the set F or not.

Representing sets by their b.r., producing a set C which is the intersection of sets A and B reduces to the question of producing a b.r. for C which has a 1 in it precisely

where both A and B have a 1 in their b.r.s. This is easily accomplished using the AND operator built into PET BASIC, which by operating on 16 bit integers, performs 15 (since we do not use the sign bit) operations at one time. Similarly the b.r. for C, where $C = A \text{ union } B$, is produced easily or ORing the b.r.s for A and B together. For c) we need in effect to create a new temporary set which contains only the element e. By then performing the intersection of this temporary with set F we produce either the null set, that is a set which contains nothing (which has the b.r. consisting of all zeroes), or a set which contains only the element e. If the result is the first alternative then e is not a member of F, if the second then it is.

A final operation sometimes encountered is that of set difference, where the set difference of two sets A and B is the set C which contains only those elements of A which are not elements of B. To produce the b.r. for C we subtract from A's b.r. the b.r. of the intersection of A and B. Pascal represents the operations of intersection, union and difference by *, + and - respectively, and we use these from now on.

Storage of the b.r.s. of sets is best accomplished within PET BASIC by the use of integer arrays, using only 15 or the 16 bits in an integer array element to hold data (the 16th bit is the sign bit, which is harder to manipulate than the others). Given that we want to manipulate N sets (we might choose $N=28$, to hold a null set, a set consisting of every element in the universe (the 'universal set') and 26 sets labelled A, B, C, ..., Z) where the universe contains M elements, we define an array SET% as follows:

```
DIM SET%(N-1,INT(M-1)/15)
```

Thus for three sets defined on the universe of 10 digits, 0..9, we would have:

```
DIM SET%(2,0)
```

The b.r. of the first set would be held in element SET%(0,0), the second set in SET%(1,0) and so on. Normally we would have more than 15 elements in the universe and hence the b.r. for a particular set would be spread over a 'column' of the array, e.g. SET%(1,0), SET%(1,1) and SET%(1,2) if there were up to 45 elements in the universe. The bit that defines whether the I'th element in the universe is in the J'th set is held in element SET%(J-1,INT((I-1)/15)) and is the $(I-1)-\text{INT}((I-1)/15)*15$ th bit in that element. Note that J runs from 1 to N and I runs from 1 to M, so that since we use the zero elements of the array, nasty expressions involving (I-1) and (J-1) crop up. If the expression SET%(J-1,INT((I-1)/15)) AND $(2^{((I-1)-\text{INT}((I-1)/15)*15)})$ yields a non-zero result then element I is a member of set J, otherwise not. Evaluating common sub-expressions will make the expression above easier to understand and faster to execute.

The listings 1 to 11 illustrate the basic operations that can be performed on the b.r.s. of sets held as described above. In all the routines it is assumed that I and J have been reduced by 1 where needed to utilise

the zero elements of the SET% array. Thus in listing 1, if we were interested in the intersection of the 5th and 9th sets in the system, I would be 4 and J 8.

Listing 12 gives a slightly forced example of the use of sets in the production of prime numbers. The program is not foolproof, since it can fail when N is too large through lack of space, or when N is too small, due to the FOR loops always executing at least once. Thus if M=0 the reference to SET%(I) in line 90 will reference SET%(1) which should not exist. Convincing examples of the usefulness of sets are difficult to find, but any book on Pascal will contain a number of attempts.

In the prime number example we knew exactly what each bit in the sieve's b.r. stood for. If it was the 0'th bit it represented the 0., if the n'th bit the number n. For more complicated examples the correspondence is usually lacking. Element 0 might be the character 'G', element 1 the number 62.5, while element 2 might represent the string 'HELLO'. To each of these elements we need assign a unique integer which represents its 'position' within the universe, or within the universal set's b.r. Similarly, when translating from a b.r. to a list of set elements we must convert from the integer 'position' within the universe to a 'value'. To do this we create a string array UNIVERSES (M-1) where the element UNIVERSES (I-1) (I runs from 1 to M) contains the string representation of the I'th element in the universe. To start off a set program the values of the M elements should be read into the array UNIVERSES and then sorted. Then when a set is defined its members can be entered and checked against the array of possible values, using for instance a binary search. If the value is not found an error message may be output, while if it found its corresponding position, and hence its unique number, will have been found. Note that a check should be made on input for an element which already exists, since by definition of a set the set $(1,2,3,1)$ is the same as the set $(1,2,3)$, or even the same as $(3,2,1)$. The unique number found can then be used in listing 7 as a value for I in order to add the element to set J.

When the values of a set are to be output in readable form a sequential search through the b.r. of the set concerned is made, and on finding a 1 in the b.r. the value of the element that the 1 represents is output, from the relevant element of UNIVERSES. Listing 13 illustrates this, formatting the result to fit on a 40 column PET screen.

As a final implementation note it is worth considering the possibility of using an array MASK% (14), where MASK%(I) contains $(2^{**}I)$, that is a one in bit position I. This will save the recalculation of the quantity $(2^{**}SB)$ which crops up a number of times in the listings (the symbol pair ** used in the article represents the up-arrow, or exponentiation sign which is not available on a typewriter keyboard).

This article, together with previous ones on sorting, searching and input, should allow the writing of a Teach Yourself Sets program, which should allow the definition of a universe, the definition of sets on that universe, and the input, evaluation and output of arbitrary set expressions. The short routine to convert from algebraic form to Reverse Polish Notation would be useful in this latter

part. Perhaps someone would like to attempt the task and get the program published in this column. It might even be of interest commercially.

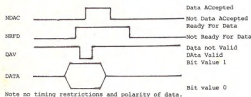
INTER-PET COMMUNICATION

Much interest has been aroused recently by various Multi-User PET systems, the basic idea being to cut down the cost of a PET based system by sharing one disk and a printer between a number of PETs. At least three systems have been on the market at one time or another, from NESTAR (of Toolkit fame), Taylor Wilson and the MU-PET system from Canada. This section of PETs corner is devoted to a minimal hardware (7 wires and two user port connectors) system for transferring data from one PET to another. While it in no way approaches the systems mentioned above it might be the answer to some problems or spark off an idea for further development.

The central idea of the system was to provide a reliable and fast method of transferring data from one PET to another. It was not to use the IEEE bus or the IEEE connector, which left only the user port. To ensure reliability a handshake system, very much like the IEEE handshake, was devised (in fact the three handshake lines are called DAV, NDAC and NRFD after the IEEE lines). However, with three handshake lines the user port no longer suffices, since there are only two independent control lines available, CA1 and CB2. The step taken was then to transfer data not one byte at a time, as in the IEEE, but rather 1 nibble, 4 bits, at a time. Using 4 bits of the user port leaves 4 for control (6 if we include CA1 and CB2), which is more than adequate for the three handshake lines proposed. In fact, should the system ever get improved it leaves room for such IEEE-like lines as ATN, EOI etc.

For each PET in the network (although the system has only been tested with two PETs there seems no reason why it should not work with more) bits 0 to 3 of the user port (mapped to location SE84F (59471)₁₀) were used to transmit the data, bit 4 acted as the DAV (Data Valid) line, bit 6 as the Not Ready For Data (NRFD) line and bit 7 as the NDAC (Not Data Accepted) line. The handshake goes just like the IEEE in that DAV, which is controlled by the transmitting (TALKER in IEEE terms) PET goes low when the data has been put on the data wires and has settled and when NRFD (controlled) by the receiving PETs (LISTENERS in IEEE terms) is set high (NOTE: as in the IEEE protocol a high line indicates the false state of that line). When NDAC (controlled by the receivers) goes high (i.e. Data Accepted) then DAV is set high again to indicate that the data on the lines will be unreliable.

The handshake is illustrated below



The general idea was tried out in BASIC to investigate the feasibility of the handshake and was successful, but at a very slow rate of data transfer, approximately 30 bits per second. The BASIC programs are reproduced below for interest.

Having proved the feasibility of the idea the programs were recoded into machine code, and the two hexadecimal dumps are also reproduced below. The transfer rate for the machine code version was somewhat faster, approximately 13000 bits per second. The machine code programs are designed solely for transfer of programs from one machine to another, and make use of page zero pointers. However the subroutines are there for a general transfer of data.

To work either the BASIC or machine code versions two user port connectors will be needed. Connect pins C to L on the bottom row of the first connector to pins C to L on the bottom row of the second connector (pin J, PA5, is not used but can be connected anyway). If anyone with access to more than two PETs tries the system please let me know if it works. Running the BASIC programs should result in whatever is typed on the transmitting PET being displayed on the receiving PET.

To use the machine code versions load the receiver and transmitter programs into their respective PETs, then do a NEW and load a BASIC program. Enter SYS(920) on the receiver PET, then SYS(932) on the transmitter. When control is returned you should find the receiver PET contains an exact copy of the program in the transmitter PET. A large BASIC STARTREK programs loads in about 11 seconds from the COMMODORE disk and about 12 from another PET via this procedure.

Details of the machine code programs:

Locations S033A to S034C in both programs hold a routine to configure the user port to the correct input/output pins. The contents of the accumulator on entry are placed in the Data Direction Register for port A. A pointer to the start of the program (assumed \$0400) is then set up, and three locations set to non-zero. These three locations will hold the last three bytes transmitted, and when all three are zero the entire program will have been transmitted (the three bytes are the zero for end of line, then a zero pointer to the next line).

Locations \$034D to \$0353 in both programs hold a routine to clear all the bits of the user port save those which have the corresponding bit in the accumulator set on entry to the subroutine.

Locations \$0354 to \$035A in both programs hold a routine to set the bits of the user port which have a corresponding bit set in the accumulator on entry.

Locations \$035B to \$0369 in both programs hold a routine which takes the byte that has just been transmitted (in the accumulator on entry) and puts it in the 'last byte transmitted' location, after moving the previous last two bytes transmitted down one byte in memory.

These three locations, representing the last three bytes transmitted, are then ORed together, so that the routine exists with the zero flag set if the program has been completely transmitted, or cleared if it has not.

Locations #036A to #036F in both programs contain a routine to wait until a bit corresponding to a set bit in the accumulator on entry, goes high on the user port, i.e. waits for a signal to go false.

In the receiver program locations #0370 to #0397 contain a routine which will receive a nibble of data over the wires, returning it in bits 0 to 3 of the accumulator. Location #0398 to #03C6 contain the main loop of the program, configuring the user port first of all, then repeating a loop which sets NDAC and DRFD low, receives a nibble and saves it, receives the next nibble and combines it with the previous one then stores the byte in BASIC program space, the checks to see if the transmission is finished after incrementing a pointer to the program, and repeats the loop if transmission has not finished. When the program has been completely received the pointer to end of program text in page zero is updated and a return from the SYS(920) executed.

In the transmitter program locations #0370 to #03A3 contain a routine to transmit a nibble of data, held in bits 0-3 of the accumulator on entry. Locations #03A4 to #03D1 contain the main loop of the transmitter program, which after configuring the user port gets a byte from the program space, sets DAV high (false) then sends the first nibble (high order) across. The second (low order) nibble is then sent and a check made for completion. If the program has not been completely transmitted the loop is repeated, otherwise return is made from the SYS(932).

Transmitter

```

033A 8D 43 E8 STA #E843
033D A0 00 LDY #00
033F 84 C0 STY #C0
0341 A9 04 LDA #04
0343 85 C1 STA #C1
0345 85 B1 STA #B1
0347 85 B2 STA #B2
0349 85 B3 STA #B3
034B EA NOP
034C 60 RTS
034D 2D 4F E8 AND #E84F
0350 8D 4F E8 STA #E84F
0353 60 RTS
0354 0D 4F E8 ORA #E84F
0357 8D 4F E8 STA #E84F
035A 60 RTS
035B A6 B2 LDY #B2
035D 86 B1 STY #B1
035F A6 B3 LDY #B3
0361 86 B2 STY #B2
0363 95 B3 STA #B3
0365 95 B2 ORA #B2

```

OPTION?

```

0367 05 B1 ORA #B1
0369 60 RTS
036A 2C 4F E8 BIT #E84F
036D F0 FB BEQ -#05 #036A
036F 60 RTS
0370 48 PHA
0371 A9 C0 LDA #C0
0373 2D 4F E8 AND #E84F
0376 C9 C0 CMP #C0
0378 EA NOP
0379 EA NOP
037A EA NOP
037B EA NOP
037C EA NOP
037D A9 D0 LDA #D0
037F 2D 4F E8 AND #E84F
0382 95 B6 STA #B6
0384 68 PLA
0385 05 B6 ORA #B6
0387 8D 4F E8 STA #E84F
038A EA NOP
038B EA NOP

```

OPTION?

```

038C EA NOP
038D A9 40 LDA #40
038F 20 6A 03 JSR #036A
0392 A9 EF LDA #EF
0394 20 4D 03 JSR #034D
0397 A9 80 LDA #80
0399 20 6A 03 JSR #036A
039C A9 10 LDA #10
039E 20 54 03 JSR #0354
03A1 A9 00 LDA #00
03A3 60 RTS
03A4 A9 1F LDA #1F
03A6 20 3A 03 JSR #033A
03A9 B1 C0 LDA #C001
03AB 66 C0 INC #C0
03AD D0 02 SNE -#02 #03B1
03AF E6 C1 INC #C1
03B1 A9 00 LDA #00
03B2 A9 10 LDA #10
03B4 20 54 03 JSR #0354
03B7 3A TBA
03B8 29 F0 AND #F0

```

OPTION?

```

03BA 4A LSR A
03BB 4A LSR A
03BC 4A LSR A
03BD 4A LSR A
03BE 20 70 03 JSR #0370
03C1 EA NOP

```

```

0302 EA      NOP
0303 9A      TXA
0304 29 0F   AND  #0F
0305 20 70 03 JSR  #0370
0309 5A      NOP
030A EA      NOP
030B 8A      TXA
030C 20 5B 03 JSR  #035B
030F D0 D8   BNE  -$20 #03A9
03D1 60      RTS

```

Receiver

```

033A 8D 43 E8 STA  #E843
033D A0 00   LDY  #00
033F 84 C0   STY  #C0
0341 A9 04   LDA  #04
0343 85 C1   STA  #C1
0345 95 B1   STA  #B1
0347 95 B2   STA  #B2
0349 95 B3   STA  #B3
034B EA      NOP
034C 60      RTS
034D 2D 4F E8 AND  #E84F
0350 8D 4F E8 STA  #E84F
0353 60      RTS
0354 0D 4F E8 ORA  #E84F
0357 8D 4F E8 STA  #E84F
035A 60      RTS
035B A6 B2   LDX  #B2
035D 86 B1   STX  #B1
035F A6 B3   LDX  #B3
0361 96 B2   STX  #B2
0363 95 B3   STA  #B3
0365 95 B2   ORA  #B2

```

```

OPTION?
0367 05 B1   ORA  #B1
0369 60      RTS
036A 2C 4F E8 BIT  #E84F
036D F0 FB   BEQ  -$05 #036A
036F 60      RTS
0370 A9 40   LDA  #40
0372 20 54 03 JSR  #0354

```

Transmitter

```

0375 A9 10   LDA  #10
0377 2C 4F E8 BIT  #E84F
037A D0 FB   BNE  -$05 #0377
037C A9 0F   LDA  #0F
037E 2D 4F E8 AND  #E84F
0381 AA      TAX
0382 A9 BF   LDA  #BF
0384 20 4D 03 JSR  #034D
0387 A9 80   LDA  #80
0389 20 54 03 JSR  #0354
038C A9 10   LDA  #10
038E 20 6A 03 JSR  #036A
0391 A9 7F   LDA  #7F
0393 20 4D 03 JSR  #034D
0396 8A      TXA

```

OPTION?

```

0397 60      RTS
0398 A9 C0   LDA  #C0
039A 20 3A 03 JSR  #033A
039D A9 3F   LDA  #3F
039F 20 4D 03 JSR  #034D
03A2 20 70 03 JSR  #0370
03A5 0A      ASL  A
03A6 0A      ASL  A
03A7 0A      ASL  A
03A8 0A      ASL  A
03A9 85 B6   STA  #B6
03AB 20 70 03 JSR  #0370
03AE 18      CLC
03AF 65 B6   ADC  #B6
03B1 91 C0   STA  (<C0>),Y
03B3 E6 C0   INC  #C0
03B5 D0 02   BNE  +$02 #03B9
03B7 E6 C1   INC  #C1
03B9 20 5B 03 JSR  #035B
03BC D0 DF   BNE  -$21 #039D
03BE A5 C0   LDA  #C0
03C0 85 2A   STA  #2A

```

OPTION?

```

03C2 A5 C1   LDA  #C1
03C4 85 2B   STA  #2B
03C6 60      RTS

```

READY.

```

10 UP=59471:REM USER PORT.
15 POKE 59459,31:REM BITS 0-4 OUTPUT, REST INPUT.
20 GET A$:IF A$="" THEN 20
30 PRINT A$:
40 BY=ASC(A$):REM BYTE TO BE HANDSHAKEN.
50 ER=0:GOSUB 1000:REM TRANSMIT.

```



```

55 IF ER<>0 THEN PRINT "ERROR:";ER
60 GOTO 20
1000 REM HANDSHAKE BYTE ON TO BUS FROM BY.
1010 POKE UP,(PEEK(UP)OR16):REM DAV HIGH.
1020 NI=((BY)AND(240))/16:REM HIGH ORDER NIBBLE.
1030 GOSUB 2000:REM HANDSHAKE NIBBLE.
1040 IF (ER<>0) THEN RETURN
1050 NI=((BY)AND(15)):REM LOW ORDER NIBBLE.
1060 GOSUB 2000:REM HANDSHAKE NIBBLE.
1070 RETURN
2000 REM HANDSHAKE NIBBLE ON TO BUS FROM NI.
2010 PD=PEEK(UP)
2020 IF ((PD)AND(192))=192 THEN ER=1:RETURN
2030 POKE UP,((PEEK(UP)AND(240))OR(NI))
2040 REM
2050 IF ((PEEK(UP)AND(64))=0) THEN 2050:REM WAIT FOR NRFD TO GO HIGH.
2060 POKE UP,PEEK(UP)AND(239):REM SET DAV LOW.
2070 IF ((PEEK(UP)AND(128))=0) THEN 2070:REM WAIT FOR NDAC TO GO HIGH.
2080 POKE UP,(PEEK(UP)OR(16)):REM DAV HIGH.
2090 RETURN
READY.

```

Receiver

```

5 UP=59471:POKE 59459,192:REM USER PORT ADDRESS, SET BITS 6-7 AS OUTPUT.
10 GOSUB 1100:REM GET A BYTE.
20 PRINT CHR$(BY):GOTO 10
1100 REM HANDSHAKE A BYTE TO BY.
1110 POKE UP,PEEK(UP)AND(63):REM NRFD AND NDAC LOW.
1120 GOSUB 2100:REM READ A NIBBLE (HIGH ORDER) TO NI.
1130 IF (ER<>0) THEN RETURN
1140 BY=NI*16
1150 GOSUB 2100:REM READ LOW ORDER NIBBLE.
1160 IF (ER<>0) THEN RETURN
1170 BY=BY+NI:REM COMPLETED BYTE.
1180 RETURN
2100 REM HANDSHAKE NIBBLE TO NI.
2110 POKE UP,PEEK(UP)OR(64):REM SET NRFD HIGH.
2120 IF (PEEK(UP)AND(16))<>0 THEN 2120:REM WAIT FOR DAV LOW.
2130 NI=PEEK(UP)AND15:REM NIBBLE.
2140 POKE UP,((PEEK(UP)AND(191))OR(128)):REM NRFD LOW, NDAC HIGH.
2150 IF ((PEEK(UP)AND(16))=0) THEN 2150:REM WAIT FOR DAV HIGH.
2160 POKE UP,PEEK(UP)AND(127):REM NDAC LOW.
2170 RETURN
READY.

```

Listing 1. Intersection of two sets.

```

1000 REM SET K=SET I INTERSECTION SET J.
1010 FOR C=0 TO N-1 : SETX(K,C)=SETX(I,C) AND SETX(J,C) : NEXT C : RETURN
READY.

```

Listing 2. Union of two sets.

```

2000 REM SET K=SET I UNION SET J.
2010 FOR C=0 TO N-1 : SETX(K,C)=SETX(I,C) OR SETX(J,C) : NEXT C : RETURN
READY.

```

Listing 3. Test for set equality.

```

3000 REM TEST FOR SET EQUALITY OF SET I AND SET J. RETURN EQUAL=-1 IF SETS
3010 REM EQUAL, EQUAL=0 IF SETS NOT EQUAL.
3020 EQUAL=-1 : REM ASSUME TRUE.
3030 FOR C=0 TO N-1
3040 IF (SETX(I,C)<>SETX(J,C)) THEN EQUAL=0 : C=N-1 : NEXT C : GOTO 3060
3050 NEXT C : REM IF WE EXIT HERE THEN SETS ARE EQUAL.
3060 RETURN
READY.

```

Listing 4. Set difference.

```

4000 REM SET K=SET I - SET J (I.E. SET DIFFERENCE).
4010 FOR C=0 TO N-1 : SETX(K,C)=SETX(I,C)-(SETX(I,C) AND SETX(J,C)) : NEXT C
4020 RETURN
READY.

```

Listing 5. Test for subset.

```

5000 REM TEST WHETHER SET I IS A SUBSET OF SET J, RETURNING SUBSET=-1 IF SO,
5010 REM ELSE RETURNING SUBSET=0.
5020 SUBSET=-1 : REM ASSUME TRUE.
5030 FOR C=0 TO N-1
5040 IF (SETX(I,C)<=SETX(J,C) AND SETX(J,C)) THEN 5060
5050 SUBSET=0 : C=N-1 : NEXT C : GOTO 5070
5060 NEXT C
5070 RETURN
READY.

```

Listing 6. Remove element i from set j.

```

6000 REM REMOVE ELEMENT I FROM SET J.
6010 SW=INT(I/15) : REM SW=SET WORD, I.E. THE WORD OF ARRAY WITHIN WHICH IS
6020 REM ELEMENT I'S BIT.
6030 SB=I-15*INT(I/15) : REM SB=SET BIT, I.E. BIT POSITION WITHIN WORD SW.
6040 SETX(J,SW)=SETX(J,SW)-(SETX(J,SW) AND (2^SB))
6050 RETURN
READY.

```

Listing 7. Add element i to set J.

```

7000 REM ADD ELEMENT I TO SET J.
7010 SW=INT(I/15) : SB=I-15*SW : REM SEE ABOVE, LINES 6010-6030.
7020 SETX(J,SW)=SETX(J,SW) OR (2^SB)
7030 RETURN
READY.

```

Listing 8. Create the universal set.

```

8000 REM CREATE THE SET I CONSISTING OF EVERY ELEMENT IN THE UNIVERSE.
8010 FOR C=0 TO INT((M-1)/15) : SETX(I,C)=22767 : NEXT C : RETURN
8020 REM MAY HAVE SOME EXTRA ELEMENTS IN THE LAST WORD OF THE B.R. UNLESS
8030 REM NUMBER OF ELEMENTS IS A MULTIPLE OF 15.
READY.

```

Listing 9. Create the null set.

```

9000 REM CREATE THE SET I AS THE NULL SET, I.E. CONTAINING NONE OF THE ELEMENTS
9010 REM OF THE UNIVERSE.
9020 FOR C=0 TO INT((M-1)/15) : SETX(I,C)=0 : NEXT C : RETURN
READY.

```

Listing 10. Test for null set.

```

1000 REM TEST THE SET I AGAINST THE NULL SET. RETURN NULL=-1 IF SET I IS NULL,
10000 REM TEST THE SET I AGAINST THE NULL SET. RETURN NULL=-1 IF SET I IS NULL,
10010 REM ELSE RETURN NULL=0.
10020 NULL=-1 : REM ASSUME TRUE.
10030 FOR C=0 TO INT((M-1)/15)
10040 IF (SETX(I,C)>0) THEN NULL=0 : C=INT((M-1)/15) : NEXT C : GOTO 10060
10050 NEXT C
10060 RETURN
READY.

```

Listing 11. Test for set membership.

```

11000 REM TEST FOR SET MEMBERSHIP. RETURN IN<0 IF ELEMENT I IS IN SET J,
11010 REM ELSE RETURN IN=0.
11020 IN=SETX(J,INT(I/15)) AND (2*(I-15*INT(I/15))) : RETURN
READY.

```

Listing 12. Prime number finder.

```

10 REM SLIGHTLY FORCED EXAMPLE OF THE USE OF SETS. TO FIND THE PRIME NUMBERS.
20 REM UP TO N (INPUT BY THE USER), USING THE SIEVE OF ERATOSTHENES.
30 REM ILLUSTRATES DATA PACKING MORE THAN SET USAGE.
40 PRINT "PRIME NUMBER FINDER." : PRINT : PRINT
50 INPUT "PRIME NUMBERS UP TO:" N : M=INT(N/15) : PRINT
60 DIM SIEVEX(M) : REM 1 SET ONLY, MAY FAIL IF N TOO LARGE.
70
80 REM PUT ALL NUMBERS INTO THE SIEVE (SET) EXCLUDING 0 AND 1.
90 SIEVEX(0)=32764 : FOR C=1 TO M : SIEVEX(C)=32767 : NEXT C
100
110 REM START SIEVING.
120 FOR E=2 TO N
130 SW=INT(E/15) : SB=E-15*SW
140 IF (SIEVEX(SW) AND (2*SB))=0 THEN 210 : REM NOT IN SIEVE=>NOT PRIME.
150 PRINT E : REM E IN SIEVE=>E PRIME. NOW REMOVE MULTIPLES OF E.
160 IF (E>(N/2)) THEN 210 : REM TO AVOID THE FOR LOOP IF NEED BE.
170 FOR I=2 TO INT(N/E)
180 SW=INT(I+E/15) : SB=I+E-15*SW
190 SIEVEX(SW)=SIEVEX(SW)-(SIEVEX(SW) AND (2*SB)) : REM REMOVE NUMBER.
200 NEXT I
210 NEXT E
220 END
READY.

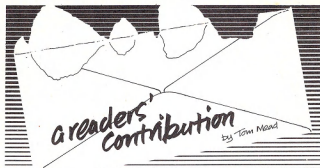
```

Listing 13. Print out the value of a set

```

12000 REM PRINT OUT THE VALUE OF SET I.
12010 PRINT "I": P=1 : NS=0 : REM NUMBER OF SETS OUTPUT SO FAR.
12020 FOR B=0 TO M-1
12030 SW=INT(B/15) : SB=B-15*SW
12040 IF (SETX(I,SW) AND (2*SB))=0 THEN 12080 : REM ELEMENT I NOT IN SET.
12050 IF (NS<0) THEN PRINT ",": P=P+1 : REM NO SPEARATOR FOR FIRST SET.
12060 NS=NS+1 : IF (P+LEN(UNIVERSE$(B)))>39 THEN P=0 : PRINT
12070 PRINT UNIVERSE$(B) : P=P+LEN(UNIVERSE$(B))
12080 NEXT B : PRINT "I" : PRINT : RETURN
READY.

```



BOMBPROOF DATA ENTRY FOR PETS

The classic input bug on the PET of aborting the program if RETURN is pressed without an entry being made has had many proposed solutions — amongst them being use of GET followed by string assembly and then conversion to numerics, poking quotes into the input buffer, and positioning the cursor over a character which is over-written on input.

There is one simple, universal, fix:

```
100 INPUT "VALUE 3 RIGHT RVS SPACE OFF 3 LEFT ";V
or
200 INPUT "NAME 3 RIGHT RVS SPACE OFF 3 LEFT ";N$
```

Either positions the cursor over a REVERSE SPACE — so that it appears to be a normal cursor. However, pressing RETURN with no entry will not bomb either input, and will return V=0 or N\$=" " (NULL) — you can then test for these values within the program — and leave the reverse space as a marker that a null entry has been made. Since discovering this method PET programming has become much less of a chore — I recommend it strongly to all users.

Incidentally, if the OFF is omitted, user input and the preceding ? print in reverse field, but the input is accepted just as if it were not reversed — e.g. RVS T is NOT delete etc., — this gives the pleasing effect of highlighting the user input.

TIRED OF QUESTION MARKS?

One thing about PET BASIC — it decides what you ought to want, and then gives it to you without the option. A case in point is the compulsory question mark supplied free with every INPUT statement. OK — I mostly do want it, but, sometimes it is not indicated and it is the very devil to remove without GET statements and associated paraphernalia. This is how you do it:

```
100 POKE 623,20 : POKE 624,20
110 POKE 625,32 : POKE 626,32
120 POKE 158,4
130 INPUT "PLEASE TYPE YOUR NAME";N$
etc.
```

Locations are for new ROMS — for old use 527+ for 623+ and 525 for 158.

How does it work — simple — CHR\$(20) is DELETE and CHR\$(32) is SPACE — put 2 x DELETE and 2 x SPACE in the keyboard buffer, tell the machine that there are four characters waiting — and rub the question mark out with them.

To use this with the debombing routine, try:

```
100 POKE 623,157 : POKE 624,157 : REM 2 BACK
110 POKE 625,32 : POKE 626,160 : REM RVS SPACE
120 POKE 158,4
130 INPUT "TYPE NAME 3 RT RVS SPACE OFF
3 LEFT ";N$
```

You can get up to other clever tricks by poking in the RVS etc., rather than printing them.

'OUT OF MEMORY' ERROR

When kids write computer programs, and start using sub-routines, they usually extricate themselves from nested GOSUBS or FOR/NEXT loops with the simple 'GOTO' instruction. Whilst the BASIC interpreter will generally disentangle the loops for them (sometimes with dire consequences — but that's another story) — each GOSUB is left on the 6502 stack until a 'RETURN' is encountered — this must be so, since otherwise recursion would be impossible. Applesoft provides a POP instruction to remove the GOSUB, and even the famed PASCAL (VCS version) provides an EXIT () instruction for clean procedure or function termination.

PET BASIC does, however, mark the position of the GOSUB return parameters on the stack by pushing the value \$8D as a marker just below them, so I cobbled together the following machine code routine to strip the return parameters from the stack:

```
033A BA TSX GET STACK POINTER
033B BD 00 01 LDA $0100,X
033E C9 8D CMP # $8D LOOK FOR MARKER
0340 F0 04 BEQ $0346
0342 E8 INX
```

```

0343 DO F6      BNE $033B   KEEP LOOKING
0345 60         RTS        NO SUB ON STACK
0346 E8         INX        GOTTIT
0347 E8         INX        SO THROW FOUR
0348 E8         INX        BYTES AWAY
0349 E8         INX
034A 9A         TXS        NEW SP VALUE
034B 60         RTS

```

Now test it:

```

10 J = 0
20 J = J+1
30 PRINT J
40 GOSUB 20

```

This should run out of memory at J = 23 — now insert

```
35 SYS 826
```

And run again — it should go on for ever.

The routine is apparently flawed, since in its stack cleanup it throws away its *own* return address — however, immediately above the subroutine parameters there is a soft return to BASIC, and the SYS 826 does not disturb the CHARGET pointer.

Be very careful with FOR/NEXT loops when combining with GOSUB — a NEXT instruction can only be correctly executed at the same level of subroutine as the FOR, and must remain dormant while at lower levels, although the loop index variable can be accessed and (may FORTRAN forgive us) altered at any level.

A SIMPLE LINE FINDER

Mainly written for practice with USR, but quite useful for amazing your friends and inserting hard to remove copy-right notices. First the machine code, with rather more documentation than usual: (NEW ROMS ONLY)

```
033A 20 D2 D6 JSR $D6D2
```

This converts the floating point number supplied by USR (X) into a two byte unsigned integer, and copies it into \$11 and \$12 in normal 6502 byte reversed form after checking that it is in range (0-65535), ready for:

```
033D 20 2C C5 JSR $C52C
```

which looks for a basic line whose number is in \$11,12. If it cannot find it it returns to the start location of the next higher numbered line and clears carry

```

0340 90 08     BCC $034A
0342 A4 5C     LDY $5C
0344 A5 5D     LDA $5D

```

The location is held as an integer in \$5C/\$5D, so load Y and A then:

```
0346 20 6D D2 JSR $D26D
```

Which converts back to floating point, then:

```
0349 60      RTS
```

If the line number is not found, then return zero:

```

034A A9 00    LDA #0
034C A8       TAY
034D F0 F7    BEQ $0346

```

(the final branch is always taken, but make it a branch rather than a 4C JMP so that the code is relocatable.

Now link in the routine by putting a JMP \$033A instruction at \$00 — 03

```
*.M 0000 4C 3A 03 .....
```

Now, write a BASIC program (or load one). If L is a line number, then:

PRINT USR (L) will print the actual location of the first byte of the line

Big deal so far — so what do we do with it? First — NEVER alter the first two bytes of the line, because they tell BASIC where the next line starts — and if you confuse it then great unhappiness will result — OK, so you are going to try it — well, save your program first. The next two are more interesting, because they contain the line number, and this can be messed about with relative impunity.

TRY

```

X = USR (L) (Make sure X is not zero!)
POKE X+2, 255 : POKE X+3, 255
LIST

```

Hey presto — you should have a line numbered 65535. POKEing with 0 will produce a line numbered zero! — even in the middle of a program!

If your copyright notice is in the middle of the program, numbered from, 1000 to 1050, then add:

```

1 FOR J = 10000 TO 1050
2 X = USR (J)
3 IF X = 0 THEN 5
4 POKE X + 2, 0 : POKE X + 3, 0
5 NEXT J
6 END

```

RUN this — and you have a copyright notice of all numbered zero in the middle of the program — very difficult to remove.

Finally, the fifth byte of the line (X+4) is the first character or token. 143 decimal is the token for REM, so

POKE X+4,143 will 'REM OUT' a line — useful for one-shot code to be executed on the first RUN but not thereafter — or disposing of a line of DATA (Thanks to RON GEERE of IPUG for this idea).

DIVERTING THE BREAK VECTOR

I was messing about with the PET input routine \$C46F which inputs a line from the keyboard and stores it in the buffer at \$0200 up, using

```
JSR $FDD0 NEW LINE
JSR $C46F GETLN
BRK FOR DEBUGGING
```

Three points emerged:

1. A count of characters input is dumped in \$A1 (undocumented as far as I know).
2. A blank line is converted to a single space.
3. The monitor grabs the first 9 bytes of the buffer on BRK to store its registers etc., and destroyed the data it was meant to be used to examine.

So, it was necessary to thwart it. BRK causes a JMI (indirect jump) to the address in \$92/93 – normally \$FD17. So change \$92 to \$3A, \$93 to \$03 so that a user routine is executed before the monitor wreaks havoc.

In this case the routine was simply:

```
033A LDX    ##$50
033C LDA    $0200,X
033F STA    $0500,X
0343 DEX
0344 BPL    $033C
0346 JMP    $FD17 – resume normal action.
```

This type of intercept has proved useful on several subsequent occasions.

D.I.Y. PET SERVICING

The Borough of Sunderland has largely standardised on the PET as a teaching micro, although some independent souls have gone their own way. The benefits of standardisation are magnified – it encourages interchange of programs and information, we can do a lot of in-house servicing and this year we ran, for the first time, a summer school in microelectronics and microcomputing – a simple recipe – take 40 bright 14 year old kids, a handful of teachers, 20 PETS, some assorted sixth formers and stir well for a week, – exhausting but very satisfying, and nothing at all to do with PET servicing. Some of the servicing tips we have sent to school technicians may be of interest to your readers:

THE UNRELIABLE KEYBOARD

The PET keyboard gradually becomes less efficient with time, particularly in the classroom environment – some keys cease to operate or require excessive force. The actual mechanism is very simple and reliable – at the base of each key there is a conductive rubber pad which presses against a gold plated PCB pattern. Over a period of time this gradually accumulates a covering of white powder which bears a strong resemblance to chalk dust or pipe ash and has insulating properties which make teflon look like a superconductor. For the penurious – this is what you do ...

1. **PULL THE MAINS PLUG OUT.** Now check that it is out.

2. Prop up the front of the PET and remove the four (two of the newer models) Philips head screws at the front underside of the cover. Put them away safely.
3. **PULL THE MAINS PLUG OUT AGAIN** – actually it is very difficult to touch a live wire if you try – but always bear Murphy's Law in mind.
4. Use the metal prop provided to keep up the lid, and identify the keyboard wires (Hint – one end of them terminates on the keyboard). Follow them down to the main PCB, where they terminate in a long pluggy thing.

Using a spirit marker, mark its orientation on both it, and the main PCB. It has, in fact, for a polarising pin to prevent incorrect insertion, but this is of little comfort if it falls out.

5. Remove the plug, and then remove the ten Philips screws holding the keyboard to the case – a real battle-ship job. Push out the keyboard assembly – this is a tight fit, but if heavy machinery is required check that you did remove all of the screws.
6. Mark the back and side of the PCB on the back of the keyboard assembly so that you can put it back the right way round – it actually will only go on one way, but it is a comforting precaution. On large keyboard pets, mark and unsolder the red and black wires to the shift lock button. Now remove the huge number of bright chrome plated screws which hold on the back. **DON'T LOSE ANY.**
7. The moment of truth. Remove the back. Don't worry about tiny springs making a bid for freedom – there aren't any. Clean down the PCB with a soft dry duster only. Don't use solvents, abrasives or chisels and be gentle.
8. Clean up the rubber pads with isopropanol (propan – 2-OL) or a proprietary record stylus cleaner on cotton buds, and keep your greasy fingers off. Keytops may be cleaned with a cloth dampened (not saturated) with a diluted detergent solution. Do not use any other solvents at all.
9. Now put it all back together again. Be *gentle*, and enter all screws loosely before tightening them.
10. Brag about it to anyone who will listen!

THE CREEPING CHIPS

Socketed IC's gradually walk out of the sockets. Blessed if I know why – probably something to do with alternate heat and cold – but walk they do. An eccentric PET often has a bad ROM contact. Cure is simple – open up the PET, and give every socketed IC a good push with your thumb. If this does not cure it, ease them out of the sockets slightly by levering gently with a flat-bladed knife and then shove them back in again. *Please* – never take them completely out – just wiggle them in and out a bit.

THE DREADED CHRISTMAS PET

One of our PET's went slightly barmy shortly after my first year sixth female students decided that it should be decorated with tinsel. I suppose I was lucky they didn't pour rum in it and set fire to it. A good size glob of BLU-TAK is very useful for picking out small metal fragments.

THE PERIPATETIC PRINTER PLUG

PET printer plugs are fitted with polarising pins to ensure they can only be put into the IEEE socket in the correct orientation. These pins tend to fall out with constant shifting of printer from one PET to another, and when eventually some four star idiot puts it upside down into the user port it burns out the 6522 VIA chip. Mark the plug and IEEE port, and preferably NEVER let kids plug it in and out. Incidentally, in the one case this happened, the only symptom shown by the PET was a refusal to save programs.

PLASTIC PETS

Some of the new all-plastic pets show a pronounced screen jitter, caused by the magnetic field from the transformer. The supplier will fit a shielding plate (FREE!) to fix this.

TEXT PRINT SUBROUTINE

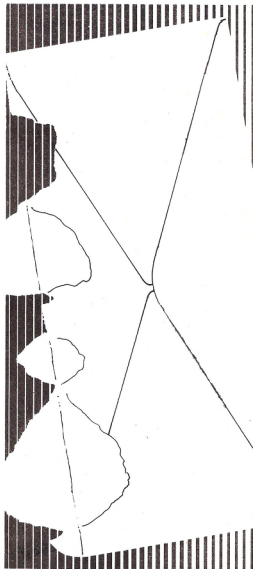
A general purpose subroutine to output text from machine code programs. The actual text is held in the main program immediately following the JSR \$033A as a sequence of hex bytes, terminated by 00.

The program as shown has its origin at \$033A, but it is completely relocatable.

```

033A BA      TSX
033B E8      INX
033C BD 00 01 LDA $0100, X
033F 85 01   STA $01
0341 E8      INX
0342 BD 00 01 LDA $0100, X
0345 85 02   STA $02
0347 A0 00   LDY $00
0349 E6 01   INC $01
034B D0 02   BNE $034F
034D E6 02   INC $02
034F B1 01   LDA ($01), Y
0351 F0 06   BEQ $0359
0353 20 D2 FF JSR $FFD2
0356 38      SEC
0357 B0 F0   BCS $0349
0359 BA      TSX
035A E8      INX
035B A5 01   LDA $01
035D 9D 00 01 STA $0100, X
0360 E8      INX
0361 A5 02   LDA $02
0363 9D 00 01 STA $0100, X
0366 60      RTS

```



The program uses zero page locations 01/02 to allow use with both old and new ROMs, but be careful — this disables the USR routine, it is best to POKE location 0 with \$60 (RTS) in case USR is invoked.

LETTERS TO THE EDITOR



Flat 11,
15 Burrot Oak Broadway,
Edgware, Middlesex.
HA8 5EJ

Dear Sirs,

You may be aware of the creation of the T.U.G. — Tangarine User Group. I am pleased to inform you that due to the increasing success of the Microtan 65 system, the group has been renamed — the Tangarine User Group International.

The aim of the group is to support, develop and evaluate Tangarine hardware and software products for the benefit of the user — whether hobbyist, businessman or researcher.

There will be an annual subscription of £5.00. Members will communicate via, a soon to be produced, newsletter which will contain equipment reviews, comments, software and anything that will interest the Microtan or 6502 user. There will also be a library of software built up and it is hoped that a circulating cassette newsletter will soon be initiated. This will enable members to communicate directly through their systems to other users, and to add their own comments or software to the end of the cassette.

Favourable rates for hardware and software products are being negotiated from various companies for user group members and some products will be available directly from the user group at a discount.

If any of your readers are interested they can contact me at the above address or write directly to the organiser:

Mr. R. Green,
3/22 Donoughmore Road,
Boscombe,
Bournemouth, Dorset.

All contributions or comments for the newsletter should be addressed to him.

Yours faithfully,
PAUL B. KAUFMAN
Asst. Editor T.U.G.I.

Dear Sir,

We have recently formed a most active club for personal computer enthusiasts and beginners who live in the West London area. We would be most grateful if you would be able to print a short news item or whatever to bring the existence of the club to the notice of those of your readers who may be interested. Please feel free to pick whatever items from the following list of features of the club you feel you may be able to include.

With many thanks for your help and co-operation.

NAME: West London Personal Computer Group.

CONTACT: Graham Brain,
81 Rydal Crescent, Perivale, Middx.
Telephone: 01-997 8986

MEETINGS: First Tuesday of each month
Willesden Technical College

FOR: Anyone, expert or beginner, owner or not,

FEATURES: Demonstrations, visits, advice, Special groups.

J. Beauchamp,
77 Grant Road,
Farlington,
Portsmouth.

Dear Sir,

Like N. KELLY in LSG 5 my APPLE suffered a currup disk directory on my 'work disk'. Faced with several long programs without recent copies I jumped at the I/O ERROR article and eventually managed to extract all the APPLE-SOFT programs, including one of 58 sectors.

There were a few problems however. The first being that if your program is longer than 4 sectors you will have to relocate the RWTS calling routine and IOB to a location where it will not get overwritten by the program. I used \$0320, this gives — POKE 812. TRACK : POKE813.

SECTOR : CALL770 : CALL800 : CALL780 etc. See listing 1.

Using these I managed to get the programs to list correctly and saved them on a good disk. Unfortunately when I tried to load the programs from the new disk I only had the first few sectors of the program. Further investigation of the CATALOG showed obviously too few sectors for each program. The problem is the End of Program pointer at location \$AF-B0 (L/H). This has to be amended to suit the program before it can be saved on the new disk. The routine CALL 780 keeps track of the high byte (\$B0), but the low byte (\$AF) will have to be changed manually. This can be done by examining memory from the monitor and knowing the format of Applesoft program lines - which is:-

location of start of next line	2 bytes	(L/H)
line number in hex	2 bytes	(L/H)
tokens		
line terminates with a null character (\$00)		

So you need to find the address of the end of the final line, and write the low order byte of this at \$AF.

In order to use all this effectively you need to know where on the disk the program is stored, this brings us back to the start of the problem - the Directory. See pages 129 to 131 of the DOS manual for a detailed explanation of its format. Probably only one sector of the directory has been damaged so it is worth trying to use RWTS to put it in RAM. POKE 812,17 : POKE813, 12 : CALL 800 . Then enter the monitor and examine \$7FF to \$87F (7FF.87F). See listing 2 for an example. Each directory entry tells you the following:-

file name
file type
file length (in sectors)
track and sector location of the file track/sector list

This last item is the important one, as this tells you the tracks and sectors (in order) of the complete program. See listing 3 for an example.

The directory continues in sector 11, then 10 etc. should the directory for the program you are really after be corrupted (as in my case), then the track/sector list can be found by a more tedious process of loading sectors into RAM and looking for the correct format of a track/sector list. Then see if the program it refers to is the one you want. The preferred sector for the track/sector list is number 12, so try these first. Obviously the sectors given by other track/sectors lists can be omitted from the search.

Having found where the program is hiding, it can be moved into RAM as before. With a program that uses more than one track, when you load the first sector from the new track then:-

POKE 812, (NEW TRACK) : POKE 813, (NEW SECTOR +1) : CALL780 : CALL800 etc.

The best advice of all is to make sure you regularly make back-up copies of everything to avoid all this hassle!

LISTING 1.

302L

```

0302- A9 07 LDA #$07
0304- 8D 31 03 STA $0331
0307- A9 08 LDA #$08
0309- 85 08 STA $B0
030B- 60 RTS
030C- CE 2D 03 DEC $032D
030F- EE 31 03 INC $0331
0312- E6 08 INC $B0
0314- A9 08 LDA #$08
0316- 8D 35 03 STA $0335
0319- 60 RTS
031A- 00 BRK

```

320L

```

3320- A9 03 LDA #$03
0322- A0 28 LDY #$28
0324- 20 D9 03 JSR $03D9
0327- 60 RTS

```

302.33E

```

0302- A9 07 8D 31 03 A9
0308- 08 85 B0 60 CE 2D 03 EE
0310- 31 03 E6 B0 A9 08 0D 35
0318- 03 60 06 00 00 00 00 00
0320- A9 03 A0 28 D9 03 60
0328- 01 60 01 00 12 08 39 03
0330- FF 07 00 00 01 00 00 60
0338- 01 00 01 EF D8 EF D8

```

LISTING 2.

7FF.8FF

```

07FF- 00
0800- 11 08 00 00 00 00 00 00
0808- 00 00 12 0C 02 C8 C5 CC
0810- CC CF A0 A0 A0 A0 A0 A0
0818- A0 A0 A0 A0 A0 A0 A0 A0
0820- A0 A0 A0 A0 A0 A0 A0 A0
0828- A0 A0 A0 04 00 15 0C 02
0830- CD CF D2 D4 C7 C1 C7 C5
0838- A0 A0 A0 A0 A0 A0 A0 A0
0840- A0 A0 A0 A0 A0 A0 A0 A0
0848- A0 A0 A0 A0 A0 A0 11 00
0850- 14 0C 02 C2 C9 CF A0 A0
0858- A0 A0 A0 A0 A0 A0 A0 A0
0860- A0 A0 A0 A0 A0 A0 A0 A0
0868- A0 A0 A0 A0 A0 A0 A0 A0
0870- A0 0C 00 17 0C 02 D1 D2
0878- C1 A0 C3 C1 CC C3 A0 A0
0880- A0 A0 A0 A0 A0 A0 A0 A0

```

```

0888- A0 A0 A0 A0 A0 A0 A0 A0
0890- A0 A0 A0 A0 08 00 18 0C
0898- 02 D0 D2 C9 CE D4 C5 D2
08A0- A0 00 C9 C3 D4 D5 D2 C5
08A8- A0 A0 A0 A0 A0 A0 A0 A0
08B0- A0 A0 A0 A0 A0 A0 A0 3A
08B8- 00 10 0C 02 D0 C1 D4 D3
08C0- A0 C2 C9 CF A0 A0 A0 A0
08C8- A0 A0 A0 A0 A0 A0 A0 A0
08D0- A0 A0 A0 A0 A0 A0 A0 A0
08D8- A0 A0 10 00 1F 0C 04 C9
08E0- AF CF A0 A0 A0 A0 A0 A0
08E8- A0 A0 A0 A0 A0 A0 A0 A0
08F0- A0 A0 A0 A0 A0 A0 A0 A0
08F8- A0 A0 A0 A0 A0 03 00 30

```

Directory continues on track 17 sector 11 track/sector
list on track 18 sector 12.

Applesoft file : HELLO
4 sectors.

track/sector list on track 21 sector 12;
Applesoft file : MORTGAGE

17 sectors.

track/sector list on track 20 sector 12.
Applesoft file : B10

12 sectors.

track/sector list on track 23 sector 12
Applesoft file : QRA CALC
8 sectors.

track/sector list on track 24 sector 12
Applesoft file : PRINTER PICTURE
58 sectors.
etc.

LISTING 3.

```
*
JPOKE812,21:POKE813,12
```

```
JCALL800
```

```
J&
```

```
*7FF,8FF
```

```

07FF- 00
0800- 00 00 00 00 00 00 00 00
0808- 00 00 00 15 08 15 0A 15
0810- 09 15 08 15 07 15 06 15
0818- 05 15 04 15 03 15 02 15
0820- 01 15 00 16 0C 16 0B 16
0828- 0A 16 09 00 00 00 00 00
0830- 00 00 00 00 00 00 00 00
0838- 00 00 00 00 00 00 00 00
0840- 00 00 00 00 00 00 00 00
0848- 00 00 00 00 00 00 00 00
0850- 00 00 00 00 00 00 00 00
0858- 00 00 00 00 00 00 00 00

```

```

0860- 00 00 00 00 00 00 00 00
0868- 00 00 00 00 00 00 00 00
0870- 00 00 00 00 00 00 00 00
0878- 00 00 00 00 00 00 00 00
0880- 00 00 00 00 00 00 00 00
0888- 00 00 00 00 00 00 00 00
0890- 00 00 00 00 00 00 00 00
0898- 00 00 00 00 00 00 00 00
08A0- 00 00 00 00 00 00 00 00
08A8- 00 00 00 00 00 00 00 00
08B0- 00 00 00 00 00 00 00 00
08B8- 00 00 00 00 00 00 00 00
08C0- 00 00 00 00 00 00 00 00
08C8- 00 00 00 00 00 00 00 00
08D0- 00 00 00 00 00 00 00 00
08D8- 00 00 00 00 00 00 00 00
08E0- 00 00 00 00 00 00 00 00
08E8- 00 00 00 00 00 00 00 00
08F0- 00 00 00 00 00 00 00 00
08F8- 00 00 00 00 00 00 00 30

```

49 Waterlow Road,
Dunstable,
Beds.

Dear Editor,

I would like to point out some transcription errors in my
Algo 68 article last issue. They are understandable because
some of the symbols used are uncommon.

The shortened form of IF a THEN b ELSE c FI was given
as (alb/c) whereas it should be (a/b/c) - verticle bars, not
ones. Similarly, the brief form of CASE clauses is (a/b ...
c/d).

Several O's were changed to C's, especially in CO
comments CO. Most of the other errors should be evident
from the context (I hope!).

Oddly enough, a 'the' seems to have been added to the
title as well!

I enjoyed reading Roger Gibson's LISP article. He made a
very good job of introducing a very unusual language, and
he made it sound good fun, which can't be a bad thing.

Can I suggest that there be an 'Under £100' section in the
L.S.G. where readers can place 'small-ads' free of charge
up to a £100 limit per ad. (Similar to the 'Under a Fiver'
schemes in many newspapers). The limit could be less than
£100 of course, and the L.S.G. may gain more readers look-
ing for a bargain or wishing to advertise their old micros
etc.

Despite the price increase, the L.S.G. is still maintaining
its position as the best value microcomputer journal, and
I'm eagerly awaiting the next issue.

Yours faithfully,

Raymond Anderson

Dear Sir,

I have been a regular subscriber to your excellent magazine since it began. We have just formed IPUG SE and wonder if you could find room for an announcement (attached of us as a User Group. I enclose a copy of our latest newsletter.

May we have your permission to reprint parts of your articles in our newsletter e.g. programmes. We will give you credit as the source, of course.

Keep up the good work.

Yours sincerely,

Mick Regus.

Independent PET Users' Group,
South East Regional Group,
164 Chesterfield Drive,
Riverhead,
Sevenoaks,
Kent. TN13 2EH

Editors Reply:-

Feel free to reprint material, so long as we get a credit.

INDEPENDANT PET USERS' GROUP - SOUTH EAST REGION

IPUGSE meets on the 3rd Thursday of each month at 7.30 pm Charles Darwin School, Jall Lane Biggin Hill. Membership includes bi-monthly newsletter 16 October 1980 PROGRAMMERS' CLINIC - Bring your programming problems - Demi of 'Supership'. 20 November 1980 DEMO HI-RESOLUTION PLOTTER & VISIBLE MUSIC MONITOR. 18 December 1980 DEMO VISICALC ON THE PET (In Orpington)

For membership write or ring:

Wing Commander M A F Ryan,
164 Chesterfield Drive,
SEVENOAKS Kent TN13 2EH

SEVENOAKS (0732) 53550

29 Valerian Close,
Chatham,
Kent.

Dear Sirs,

I enclose a short article plus two listings to solve what I suspect must be a common problem with Commodore disk drives. As far as I know there is no published solution to the unclosed disk file, and nobody I spoke to had an inkling.

I hope you will be able to publish it. I am sending it to you exclusively as I think it is above the nursery-level gamester readers of the monthly glossies! (If it hasn't got graphics reject it is their policy)

Yours ever,

Alan Walters.

THE PROBLEM OF THE UNCLOSED DISK FILE ON THE COMMODORE DISK DRIVE AND A SOLUTION BY ALAN WALTERS.

Users of the Commodore dual drive floppy disk unit (Models 2040 and 3040) will already be aware that unless a sequential file is closed immediately after writing it is not possible to access it sequentially subsequently. Attempts to open the file result in error message WRITE FILE OPEN. It is not possible to either close or open it. This is because the Block Availability Map was not updated by CLOSE, so there is no record upon the disk of the number of sectors used, or the location of the disk file. Listing the directory shows zero blocks used by the file and a star in front of the SEQ description.

All, however, is not lost. Although the BAM is not aware of it the data did get written to the disk, and is all there with the possible exception of the final sector, which would have been written upon CLOSE. We can get to this information, pull it, and write it away to another sequential file. (remembering this time to close it!) The secret lies in knowing where the file begins on the disk, and then following it through until we find its end. Trial and error using direct-access methods is the only way, but we are faced with three levels of difficulty:

1: Easy: the file was written to a clean disk, so is the only data upon it. The file will start at track 17, block 0, work downwards to track 1, jump to track 19, then work upwards track 35.

2: Hard: the file was written to a disk with other files already on it. The file will start somewhere on the disk (you can get a rough idea by looking at the number of blocks used by the other files) but should then follow the same rules as 1, above.

3: Forget it: at some point in the past a program or file smaller than our new file has been scratched. Our new file will first fill the space originally used by the old file then jump around more recent files to find itself more room. This facility of the drive is what lets us start a program, save it write other programs and save them, then go back to our first program, finish it and re-save it - it slots in wherever it can. When loading such a program you may hear the head swinging back and forth on the disk. The answer here is just trial and error plus a certain amount of guesswork to make a 'map' of the file. In all the above cases sector order is simpler. Assuming we start at track 17, sector 0, the next sector within 17 that we have to look at is given by the second byte in sector 0. Each successive sector number is given by byte 2 of the previous block. There is no hard and fast rule to this - it seems to vary with sectors in a track. Track 17, for example, goes in this order: 0, 10, 20, 8, 18, 6, 16, 4, 14, 2, 12, 1, 11, 3, 13, 5, 15, 7, 17, 9, 19. Bytes 3 to 255 (ie, the rest) is the data. There is nothing, however, to tell us when to switch tracks so we have to keep a count of sectors-read, then switch when the count equals sectors in the block.

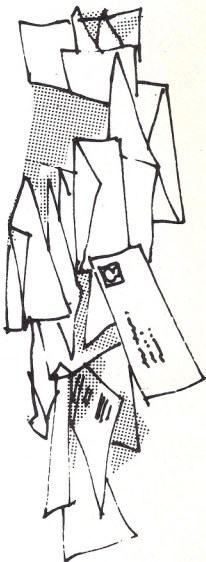
Program 1 is a simple direct-access method of looking at a specified block in either literal or ASC values, to be used in the trial and error stage. Program 2 is an example of how the information can be then pulled and rewritten to

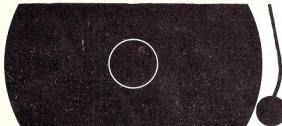
another disk. Starting and ending sector numbers are built in and may be tailored according to whatever program 1 tells you. The old file must be in drive 1 and a clean disk in drive 0. "U1" is a replacement for "B-R" but gives access to the whole block at once via GET.

Perhaps it is obvious, but it should be pointed out that the blocks used by the old file are not 'allocated', ie, not reserved, so any writes to that disk may well overwrite the file. Due to this it is advisable to do nothing to the disk except initialise it and read it. The CLOSEs in program 1 do not cause any writing to the disk as far as I know.

```
0  REM PROGRAM 1
5  OPEN1.8.15: OPEN2.8.4. "##": GET#2.A$
40 PRINT: INPUT "TRACK,SECTOR,1 FOR ALPH
"; T. S.A: IFT=99 THEN CLOSE2: CLOSE1: END
50 PRINT #1, "U1"; 4; 1; T; S
70 GET #2, A$: Q=ST
75 IFA$=" " THEN A$=CHR$(0)
77 IFA=1 THEN 100
80 PRINTASC(A$);: IFQ 64 THEN 70
90 GOTO40
100 GOTO40
READY
```

```
0  PROGRAM 2
1  DIMT (35.2): FORX = 1 TO 35: T(X.1)=X: READ T
(X.2): NEXT
2  DATA 21.21.21.21.21.21.21.21.21.21.
21.21.21.21.21.20.20.20.20.20.20.20
3  DATA 18.18.18.18.18.18.17.17.17.17.17
5  OPEN1.8.15: OPEN2.8.4. "##": GET#2, A$
10 OPENS8.8.8. "0: NEWFILE.S.W."
40 T=17: S=0
50 PRINT #1, "U1"; 4; 1; T; S
70 GET #2, Z$: GET #2, N$: IFN$=" " THEN N$=
CHR$(0)
71 R = R + 1
72 GET #2, A$: Q = ST
75 IFA$=" " THEN A$=CHR$(0)
80 PRINT #8.A$: IFQ 64 THEN 72
85 IFT = 23 AND S = 6 THEN 210
90 S=ASC(N$): IFR=T(T.2) THEN R=0: GOTO 110
100 GOTO 50
110 IFT=0 THEN T=19: GOTO 50
120 IFT = 18 THEN T=T+1: GOTO 50
125 T=T-1
130 GOTO 50
210 CLOSE8
220 CLOSE2
230 CLOSE1
240 END
READY
```





lan Thompson

Alarming your computer

Alarming your computer.

It goes without saying that the Video Genie and TRS-80 level II computers probably support the most advance BASIC interpreter in a computer system for under a £1000 even though these units are less than £500.

But they do lack the facility to attract the operators attention by means of sound. I hope to remedy this slight defect in this artical using a small amount of hardware and a software patch to drive people crazy.

Video Genie Hardware.

The hardware for the Video Genie is a simple matter of a peizo buzzer similar to the one TANDY sell Cat. No. 273-060 and a piece of wire.

First open the video Genie case by undoing the screws on the bottom of the case. When you have removed the top of the case unscrew the keyboard and move to one side taking care not to disconnect the connector (if you do I wish you the best of luck putting it back in!). You now

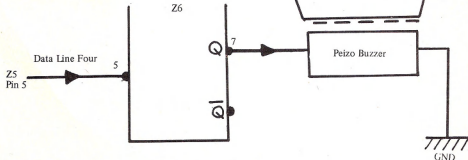
have access to the two circuit boards, locate chips Z6 and Z5 on the interface board, this is the board nearest the power supply, using Figure 3.

Solder a short piece of wire between pin 5 Z5 to pin 5 Z6, and connect the red or positive lead on the peizo buzzer to pin 7 Z6 and the black lead or negative lead to pin 8 Z6. The best way to mount the peizo buzzer is to use some double sided sticky foam and attach the buzzer to the underside of the circuit board. Now all that remains is to put the computer back together again.

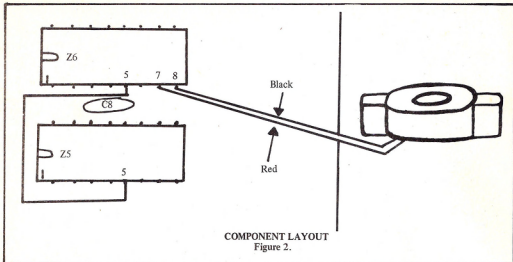
TRS-80 Hardware.

For this machine we require a little more hardware, 3 pieces of wire, 1 74Ls175 and the peizo buzzer.

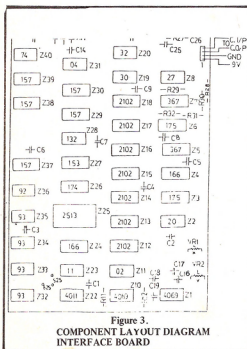
First pull the TRS-80 apart by undoing the 6 screws on the underside of the machine and then lay the main circuit board and keyboard flat on a surface with the component



CIRCUIT DIAGRAM
Figure 1.



side up. Carefully disconnect the level 11 board flat cable connector.



Bend all the pins out except for the 4 corner pins on the 74LS175 as shown in figure 4. Piggy back this chip to Z59 on the circuit board see figure 5 for location, using the

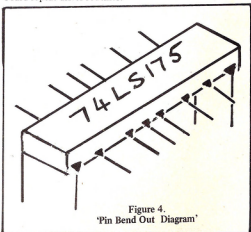
same orientation. Connect pin 9 Z60 to pin 5 to ZNEW and connect red or positive lead of the peizo buzzer to pin 7 ZNEW using a piece of wire and do the same for the negative lead connecting it to pin 8 ZNEW. The best place to mount the buzzer is between the LED and the space bar on the keyboard. Thus the job is completed.

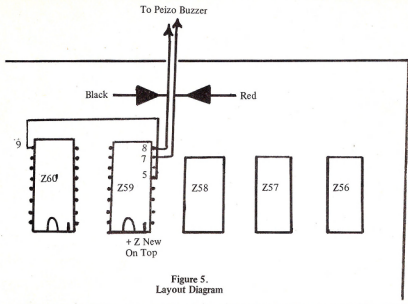
Testing.

The easiest way to test the unit is to poke 16445,16. This should result in a loud signal of about 4KHz.

Software Patch.

The hex listing given at the end of this article should be saved on tape using an execution address of 7EE8H. The program is located in this area so the top page of memory in a 16K machine is free for lower case or automatic keyboard repeat driver routines.





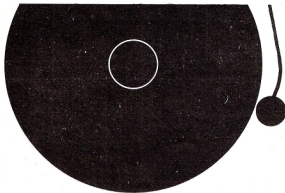
The software generates the TRS-80 concerto every time an error is printed on the screen or when the basic interpreter encounters an NAME instruction.

100 NAME: PRINT
"JOB FINISHED"

To get the patch to work when routine has been loaded execute it by typing '/32448' under the SYSTEM mode.

HEX LISTING.

7E90: 00	F5	15	D5	C5	21	C9	7E	4E	79	87	28	12	25	46	5A
7EA0: 3D	40	C0	E7	D3	FF	32	3D	40	10	FE	46	3A	3D	40	CB
7EB0: A7	D3	FF	32	3D	40	10	FE	0D	20	E3	23	10	FE	10	FE
7EC0: 10	FE	18	D4	C1	D1	E1	F1	C9	A0	90	3F	A2	5C	AC	60
7ED0: 90	A0	90	40	90	70	80	F0	90	5D	A2	5B	AD	60	90	ED
7EE0: 6B	48	5F	FF	54	00	00	00	21	91	7E	22	A7	41	22	8F
7EF0: 41	3E	C3	32	A6	41	32	8E	41	C3	19	1A	00	00	00	00



Apple Pips

N. Kelly



'Since I first listed through Applesoft's ROMs, I have been both fascinated and bewildered by what I found.

Recently, much information has been published in 'The Apple Orchard' by J. Crosley, supplemented by R.M. Mottolla in 'Micro-6502, August 80'.

My particular interest is the floating-point package. This is based on a pair of extended six-byte accumulators in 'Page Zero', called 'FAC' and 'ARG'. These are supplemented by several five-byte temporary stores, and all the named math variables in memory. The routines use the format, (ARG @ FAC) with normal arithmetic precedence, leaving the result in FAC.

At first, I tried to use the package like a programmable calculator. 'Put this here, that there, add, subtract, multiply, divide.' It didn't work. Routines would stop, change numbers in mid-run, or invent them. One formula would work the first time through, but not again. It puzzled me.

The clue lay in the multiple entry points for each basic operation. For an example, I'll use 'Divide';

FDIV EA69 (ARG/FAC), result in FAC.

FDIVT EA66 Load ARG from pointers, fall into FDIV. These came from 'Apple Orchard', but a month later I read of,

FDIV2 EA60 Load FAC from pointers, jump into FDIV. Along the way, these two set flags and pointers to ensure FDIV would work. Some tell the Apple 'This is not a string'. One checks the FAC exponent has been copied to the 6502's 'A' reg. This latter explains the occasional, other-wise inexplicable, 'LDA £9D'.

The listing (Figure 1) is an example of how some of the routines may be linked. I have employed several numbers from within the ROMs, as I can't yet persuade the machine to locate named variables.

Please note that any attempt to use 'JSR ED2E' within USR () will cause an error message. Also, this subroutine will totally scramble the FAC during its use, hence its use last.

I have indicated with 'NOP's where a 'JSR ED2E & RTS' would show progress. If you use the mini-assembler, re-

member to reset the ROMs to Applesoft before '300 G'. P.S. As far as I can tell, the answer's 1.

```

0300 A9 01 LDA #£01
0302 20 93 EB JSR £EB93 ; FLOAT
0305 EA NOP
0306 20 63 EB JSR £EB63 ; COPY FAC TO ARG
0309 EA NOP
030A A5 9D LDA £9D ; PATCH FAC
030C EA' NOP ; FOR RE-USE IN
030D 20 C1 E7 JSR £E7C1 ; ADD
0310 EA NOP
0311 A9 50 LDA #£50 ; LO ) # 10
0313 A0 EA LDY #£EA ; HI )
0315 20 66 EA JSR £EA66 ; LOAD ARG, DIVIDE
0318 EA NOP
0319 20 D0 EE JSR £EED0 ; NEGOP, F = -F
031C EA NOP
031D 20 A0 E7 JSR £E7A0 ; F = F + .5
0320 EA NOP
0321 20 23 EC JSR £EC23 ; INT
0324 EA NOP
0325 20 AF EB JSR £EBAF ; ABS
0328 EA NOP
0329 A9 13 LDA #£13 ; LO ) # 1
032B A0 E9 LDY #£E9 ; HI )
032D 20 BE E7 JSR £E7BE ; LOAD ARG, ADD
0330 EA NOP
0331 A9 64 LDA #£64 ; LO ) # ¼
0333 A0 EE LDY #£EE ; HI )
0335 20 7F E9 JSR £E97F ; LOAD ARG, MULTIPLY
0338 EA NOP
0339 A9 6B LDA #£6B ; LO ) # 2 TT
033B A0 F0 LDY #£F0 ; HI )
033D 20 A7 E7 JSR £E7A7 ; LOAD ARG, SUBTRACT
0340 EA NOP
0341 20 8D EE JSR £EE8D ; SQR
0344 EA NOP

```

```

0345 20 63 EB JSR  $EB63 ; COPY FAC TO ARG
0348 EA      NOP    ; BUT NO PATCH AS
0349 A9 37 LDA     # $37 ; LO ) # - 1/2
034B A0 E9 LDY     # $E9 ; HI )
034D 20 F9 EA JSR  $EAF9 ; MEM TO FAC
0350 EA      NOP
0351 20 97 EE JSR  $EE97 ; ARG FAC
0354 EA      NOP
0355 20 90 EB JSR  $EB90 ; SGN
0358 EA      NOP
0359 A2 66 LDX     # $66 ; LO )
035B A0 03 LDY     # $03 ; HI )
035D 20 2B EB JSR  $EB2B ; FAC TO MEM
0360 EA      NOP
0361 20 2E ED JSR  $ED2E ; PRINT FAC
0364 60      RTS    ; BUT CONTENTS LOST!
0365 EA      NOP
0366 EA      NOP      ; START OF MEM
0367 EA      NOP      ; FOR JSR EB2B
0368 EA      NOP
0369 EA      NOP
036A EA      NOP      ; (5 BYTES USED )
036B EA      NOP
036C EA      NOP

```

END

```

MM" ;: NORMAL
1025 INVERSE : PRINT "D" ;: NORMAL
      : REM = CTRL D
1030 PRINT "SAVE FOR EVEN FOULER TRICKS !! " ;
1035 INVERSE : PRINT "MMD" ;: NORMAL
1040 PRINT "LOCK FOR EVEN FOULER TRICKS !! "
1050 REM

```

LINE 0 DOES NOT 'LIST' AS SEEN ABOVE. TO ENTER IT CORRECTLY,
FIRST ' RUN 1000 ' THEN MOVE
THE CURSOR ONTO THE '0' AND
COPY THE APPARENT GIBBERISH AS
A NEW LINE. NOW, TRY TO LIST !!

```

1060 REM
PS. FOR EASY REM FORMATTING,
' POKE 33, 33 ' THEN HAVE A PAIR
OF 'INVERSE-M'S AT THE START
OF EACH NEW 'LINE'. USE: -
      INVERSE ; ? "M" ;: NORMAL
TO MAKE THEM AS NEEDED.

```

2000 END

2500 REM

AND AGAIN

3000 GOTO 4999:

OR HERE

4999 END:

OR HERE

5000 PR # 0

5010 IN # 0

5020 REM

'RUN 5000' TO DISCONNECT DOS.
'CALL 1002' TO RESUME,

IMPOSSIBLE

Will your APPLE give the underline character direct from the key-board? Try this;

Holding down 'SHIFT', depress both 'U' and 'I' together, then also depress 'Y'.

This gives me "Y-" or "-Y". That's right, one (?) key-press gives two characters!

Repeating for 'JKLNM' sometimes gives the Back-slash, square brackets and caret. Also, whatever symbol came last will obey the 'REPEAT'.

Keyboards differ. If yours does not display this quirk, try other combinations. I discovered mine by accident, after all!

SOME FOULER TRICKS

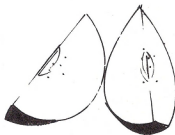
```

999  END
1000 PRINT : PRINT "0 REM" ;: INVERSE
      : PRINT "HHHHHHH" ;: NORMAL
1003 PRINT "      " ;: REM 8
1005 INVERSE : PRINT "MMM" ;: NORMAL

1010 PRINT " ( YOUR MESSAGE ETC.)
      " ;
1013 INVERSE : PRINT "MMM" ;: NORMAL

1015 PRINT " ( AND MORE OF THE SA ME !! ) " ;
1020 INVERSE : PRINT "MMMMMGGGGG

```





THE ROMPLUS+ AND KEYBOARD FILTER A USERS REVIEW

By Rob Benyon

Department of Biochemistry
University of Liverpool P.O.Box 147 Liverpool L69 3BX

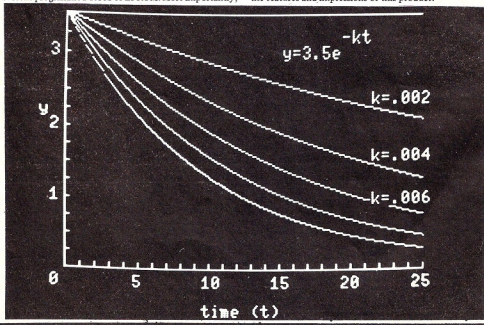
Introduction

A major deficiency in the otherwise impressive specification of the Apple II was the inability to mix text and graphics on the same page of display. At any time one could display either text, low resolution graphics or high resolution graphics, but never any mixture of the three. The ability to generate high-resolution shapes on the high resolution display meant that characters could be defined as a series of vectors but drawing them in, the form of, for example, a sentence was still a protracted programming exercise. Last year saw the appearance of a new peripheral board for the Apple II from Mountain Hardware Inc., California. This product was the ROMPLUS+, a plug in board that allowed the user to call individual programs resident in commercial or custom-programmed ROM or EPROM. More importantly,

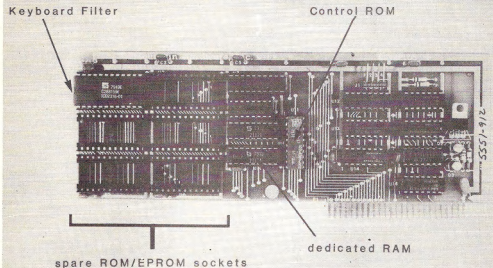
the ROMPLUS+ was provided with one piece of firmware installed — the Keyboard Filter. Among the features of the Keyboard Filter are the following:

1. The display of upper and lower case letters.
2. Mixed text and high-resolution graphics.
3. Single-key macro commands.
4. Multiple character fonts.
5. For users without the Autostart ROM-improved editing features.

From this list alone it is apparent that the Keyboard Filter extends the capabilities of the Apple II to a considerable degree. This brief review will attempt to describe some of the features and impressions of this product.



The ROMPLUS+ with Keyboard Filter installed



The ROMPLUS+ with the Keyboard Filter installed (top left socket). The board also contains 256 bytes of RAM for use by user ROMs and has a control ROM.

Installation

The ROMPLUS+ is provided with the Keyboard Filter installed (fig 1) and with a disk containing demonstration and utility programs. The card may be plugged into any slot except #0 and is turned on in the same way as any other peripheral — using the "PR ## (slot number)" command. An elegant touch was the provision of a short program that, when executed, finds which slot contains the ROMPLUS+ — making programs that call this peripheral slot — independent. This program is also supplied as a text file that may be EXEC'd into a program.

graphics and multiple fonts all available. For normal text output the 8k graphics screen behaves like 1k text screen of 24 x 40 characters. However, graphics lines, shapes etc, may be drawn on the same page — permitting annotated graphics and figures. Attempting to write text below the bottom line of the screen causes scrolling of graphics as well as text. Listing of programs is slower because each screen is being written in 8k of memory — this can make following a listing much easier.

Multiple fonts

The Keyboard Filter allows the definition of an unlimited number of different character sets, defined in a 7 x 8 matrix and each character is stored as a bit pattern in 8 bytes. Thus, a full 256 character set occupies 1k of memory which is loaded at a default of \$ 6000 (fig 2). A set of programs are provided on disk which allow the definition of new character fonts. Several fonts are provided as binary files on disk, including upside down font, man font (which defines a man who is made to run across the screen in one of the demonstration programs!) and a number of others. The font editor program is very easy to use.

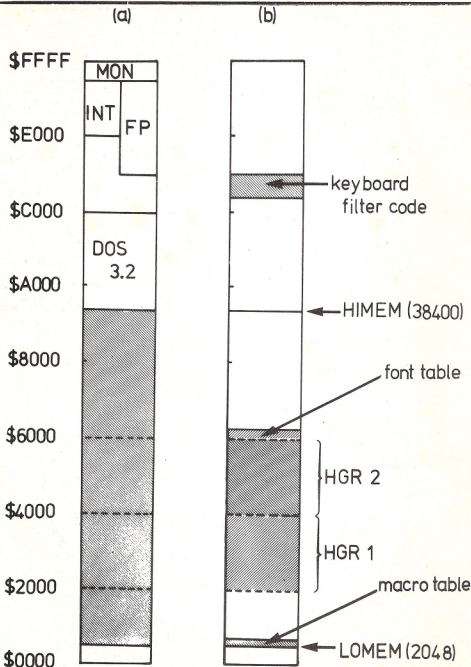
Switching between fonts is very easy, using the 'control F n' command where n is the number of the font required. The characters may then be incorporated into print statements or used with the CHR\$(n) command.

Initialisation

Several initialisation options are available with the Keyboard Filter:

- A. First time with graphics
- B. Re-initialisation with graphics
- C. First time text only
- D. Re-initialisation text only

If the graphics option is chosen (A or B) then the screen display is mapped by one of the two-hi-res graphics pages (fig 2), each occupying 8k bytes. Text mode uses the normal text page and lower case or graphics is not available. We have generally found the graphics pages to be the most useful — as we have upper/lower case, mixed text and



Memory map of the Apple II showing areas used by the Keyboard Filter. Abbreviations FP, - Applesoft, INT-Integer Basic, MON;-monitor. The diagram (a) shows the area of RAM normally available to the user. Diagram (b) shows the parts of memory that can be used by the Keyboard Filter.

Keyboard macros

The ability to define Keyboard macros is due to the constant surveillance of Keyboard input by the Keyboard Filter. A "controls", when encountered at any time except during LISTING, will cause the Keyboard Filter to wait for a second key. When this key is pressed a command string associated with that key is immediately executed. For example "control S" P might cause the execution of:

```
"HOME: POKE 33, 33: LIST : POKE 33, 40"
```

for editing lines containing text in strings.

Again, a macro editor is provided on disk which is very easy to use. Each command string may be up to 63 characters long and may include carriage returns. Multiple macrotables may exist, but only one, loaded at \$8000, is active at any time (fig 2).

Memory conflicts

The ROMPLUS+ and Keyboard Filter act as peripherals and use the \$C800 - \$CFFE address space. Consequently, it cannot be used at the same time as any other peripheral that uses this space (reserved for control ROM's), because only one peripheral can be active at one time. This means that the Mountain Hardware Clock and the Apple High speed serial card cannot be called through the Keyboard Filter - no solution to this problem is given. Other

peripherals, such as the parallel interface card can pass characters through the Keyboard Filter and may be active at the same time as the ROMPLUS+.

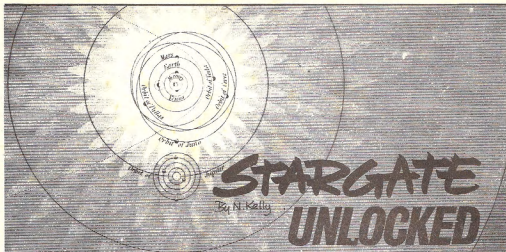
Other memory problems are likely to arise from the use of the 8k high resolution pages. The space from \$8000 to \$20000 (6k bytes) may not be large enough to take your program and it may spill into HGRI (\$20000 - \$3FF). Erasing this screen space will also erase part of the program (CRASH!). The solution to this is to move the bottom of your program (LOMEM) to \$4000, above HGRI - leaving 22k bytes free before crunching into the bottom DOS 3.2. No problem should arise from the placement of the font or macrotables as they contain no address sensitive information and may be relocated at will.

Conclusions

In general, a very impressive piece of hardware that we shall put to good use once all of programs are written to use it as a matter of course. I hear rumours (albeit unsubstantiated) that the Keyboard Filter is incompatible with Pascal. A minor niggle - the demonstration programs were all in Integer Basic - with the number of Apple II plus models around there must be a lot of Applesoft users in circulation.

Finally, if anyone would like to see the ROMPLUS+/Keyboard Filter in action or discuss it further, I'll be pleased to help - my telephone number is 051-709-6022 ext. 2757.





The publication of 'Stargate - A 3D planetarium' in the May 1980 edition of the Software Gazette has prompted a number of enquiries. Most have asked, 'But how does it work?' 'How do I get a good view?' or 'How may I adapt it to Pet/Tandy/ITT 2020?'. Some hints follow.

In 1978, I wanted a map of the nearby stars. A hunt in Liverpool's Central Library found ample information, but I had to interpret an unfamiliar system. Astronomers traditionally chart the heavens by Right Ascension, Declination and parsecs. These correspond to longitude, latitude and distance. The Vernal Equinox replaces Greenwich as the Prime Meridian, but equator and poles are common to both systems.

Initially, I tried to obtain matching radially-ruled graph paper. I hoped to plot R.A. 'around', and distance $\times \cos(\text{Dec})$ out to give a map. A note of distance $\times \sin(\text{Dec})$ against each point would give the height. From these, it would be easy to calculate interstellar gaps. Unfortunately, I could only get square-ruled paper, so I had to convert to XYZ format.

The two systems are related to the Stargate display as shown in Fig. (1). With the observer facing forwards, towards the Vernal Equinox, Right Ascension increases anti-clockwise around his feet. 'Up' and 'Down' may be imagined as facing forwards, then tilting the neck.

This viewpoint may be shifted to any position in the permitted cube. The 'Depth of View' may be set to exclude background stars, the display panned, tilted and zoomed to 32x magnification. The stars are correctly dimmed with distance, or protected from loss by the optional automatic brightness control 'C'.

At this point, the three modes, 'MAP', 'VIEW' and 'TANK' part company. MAP remains a graph-plot, an architect's plan or elevation, with sight lines parallel. VIEW simulates the 'actual' appearance of that piece of sky, and represents a moveable window in space. By dividing distance across by distance out, it produces a conical field of view, whose sight lines meet at the eye. This mode is best suited to Stargate 600, when the famous constellations may be seen.

TANK is different. This mode has a vanishing point at 100 light-years, and gives a true perspective on the stars. Consider a glass table in a dark room; Scatter stars in the air, draw luminous verticals from stars to table, then connect these with radials to the given TANK centre, fig. (2).

For the best illusion of 3D, your viewpoint must be removed from the TANK centre and the plane of the table. This may be done by XYZ shifts, TANK centre offsets, pan, tilt and combinations of these. ZOOM plays an important role. Too small, the stars blur together or perspective is lost. Too large, the field of view shrinks from interest or, worse, the TANK centre goes off-screen and Stargate complains.

LOOK F AT ZOOM 9

MODE IS TANK FROM -15, 0, 4

CENTRE 0, 0, -3 @ 0, then with ABC.

JUMP 'locates selected stars, predicts and plots interstellar routes'. Every starship should have one. It begins by asking for a starting point and a destination. Best used with TANK, it blinks the end-points of your proposed jump, quoting the true interval to help you decide. If approved, the track is drawn and your position pointer advanced, ready to continue.

JUMP reports and excludes any destination that lies off-screen or beyond the 'depth of view'. It will display stars that were invisible because the automatic brightness control 'C' was off. Once queried, such stars remain lit.

As an example of TANK, see JUMP, try;

LOOK L AT ZOOM 15 WITH ABC

LOOK B AT ZOOM 14 WITH ABC

MODE IS TANK FROM 1, 10, -5

MODE IS TANK FROM 15, 3, -4

CENTRE 2, 12, -10 @ 0

CENTRE -6, 4, -8 @ 0

Then take a trip ;

Sol (0) to	Proxima Centauri	(37)
	Alpha Centauri	(39)
	Epsilon Iota	(64)
	CD -36 15693	(69)
	UV Ceti (L726.8)	(6)
	Tau Ceti	(7)
	Epsilon Eridani	(10)
	Sirius	(18)

and back to Sol, Stargate no (0).
Just 49 light-years ;

For those with Stargate 600, may I suggest the following views;

Pegasus	Look F @ Zoom 10
Orion, Taurus	Look L @ Zoom 16
Virgo, Corvus	Look B @ Zoom 16
Hercules	Look R @ Zoom 10
Plough/Dipper	Look U @ Zoom 10

I find the 'NORTON'S STAR ATLAS' an excellent aid.

Stargate 100 runs on any Apple or ITT 2020 with 32k and Applesoft. If you do not have a disc-drive, don't despair. First, write a short program to dimension the array and allow entry and editing of the data. Then, off-line, use 'STORE A%' to dump the array onto tape. In triplicate, on two tapes, please! Run Stargate. When it halts, pleading, 'Array is empty', type 'RECALL A%', then 'GOTO 300'.

This may sound risky, but it is not. When editing Stargate, I would reload the array after every alteration, and rarely had to try twice. I used a 'jack to DIN' lead and the TAPE socket of my recorder to bypass volume and tone controls, and allow me to monitor the transfer.

To aid conversion to another machine, please take note of the following quirks of Applesoft.

1) IF / THEN statements default to the next *line*, not the

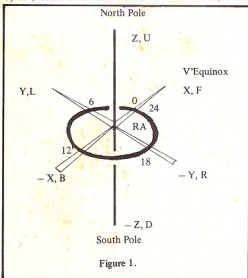


Figure 1.

next statement on the IF's line.

- The Apple hi-res screen has an origin in the top left corner, with X = 0 to 279 across, and Y = 0 to 159 down. The four lines of text at the bottom are independent of the graphics, and scroll within their own window.
- GET is a mini-INPUT. It halts the machine, and waits for a key-press.
- PEEK (-16384) polls the keyboard. If this location goes 'high', then a key was pressed. A subsequent GET statement will retrieve that key-stroke without halting the machine.
- String handling; Applesoft sets a limit of 255 characters per string. A statement such as 'DIM A\$(4)' opens a five-string array, each of the maximum length.
- MID \$(A\$, X, Y) returns Y characters, beginning at position X of the string.

Best of luck!

N. KELLY BSc AFBIS.

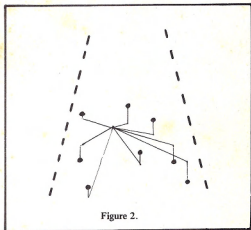
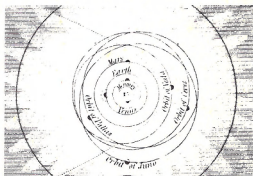


Figure 2.





Tangerine article

by P.B. Kaufman

This new 6502 based system is available in Kit Form with each module purchased separately or as a complete packaged unit, known as the 'MICRON', offering a full ASCII keyboard, video display, 10K extended BASIC, 8K of RAM and cassette file handling.

The main CPU board, the MICROTAN-65, consists of a very high quality video display section, 1K of RAM, 1K of ROM which contains the monitor program TANBUG, and space for a lower case option and a chunky graphics option. The board is double sided through-hole plated and makes construction very straightforward, you can have it up and running in the same evening. Pricewise, the MICROTAN-65 is very competitive, and offers a lot more than its main rivals, the ACORN and the KIM.

The 1K RAM is used by the monitor and stack, and the VDU memory leaving just under 1/2K for user programs, several of which are given in the excellent Tangerine manual.

Tanbug, the monitor program offers a large number of facilities for machine code programming. Its full repertoire is listed in Table 1. Tanbug will run with either the basic hex-keyboard or a full ASCII keyboard, it works out itself which type is being used. If the expansion board, TANEX is used there is an expansion to TANBUG known as XBUG which plugs into it and contains cassette file handling, an assembler, and disassembler. There are two cassette speeds available, a CUT's speed at 300 baud which gives high reliability at the expense of more tape and a 2400 baud 'TANGERINE' format which gives higher speed but is very sensitive to dropout and other tape problems.

TANEX offers 7K of RAM expansion, 6K of ROM 32 parallel I/O lines, two TTL serial I/O ports, a RS232 I/O port, 4 16-bit counter/timers and 10K Microsoft BASIC. The timers and parallel lines are configured using two 6522 VIA chips. The 6K ROM includes the 2K of XBUG and room for TANDOS - a 4K disc operating system. The BASIC is the first implementation of Microsoft's 10K version, which includes full array and string handling, READ and DATA statements and direct interrupt handling.

Further expansion includes TANRAM which provides 48K RAM, a high density graphics board, PROM programmer and A-D, D-A boards.

All in all the Tangerine Microtan-65 offers a low-cost entry into computers and programming with easy expansion into a complete and professional microcomputing system.

Mxxxx	examine or modify address xxxx
If	close present location and display next one.
esc	close present location and display previous one.
cr	close present location.
space	re-open present location, ignore any previous change.
R	examine pseudo register locations in memory
Gxxx	begin user programme execution at location xxxx
S	initiate single-step mode.
N	normal mode, clears single-step.
Lxxxx, y	list memory from xxxx, y lines of 8 bytes.
Cxxxx, yyyy, zzzz	copy memory block xxxx - yyyy to locations, starting at address zzzz.
Oxxxx, yyyy	calculate hex offset between location xxxx and yyyy (used for branch instructions).
Bz, xxxx	set breakpoint number z at location xxxx.
B	clear all breakpoints.
P(x)	Proceed past breakpoint(s), or in single step mode proceed through next x instructions.

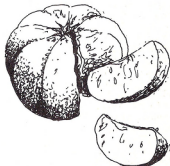
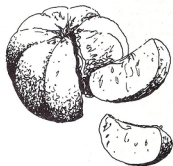
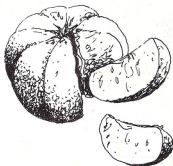
XBUG COMMANDS

C	set CUTS cassette speed - 300 baud.
F	set Fast cassette speed - 2400 baud.
Dxxxx, yyyy, N	dump locations xxxx to yyyy to cassette using file name N (max 8 characters).
F, N	fetch File N from cassette.
E, N	verify File N with memory, also used to give a directory of contents of tape.
Txxxx	enter line-by-line Translator (assembler) at location xxxx.
Ixxxx	Interpret (disassemble) instructions starting at location xxxx

Table 1: TANBUG's and XBUG's repertoire.

FFFF	TABBUG monitor
FC00	
FBFF	(TANBUG REFLECTED)
F800	
F7FF	XBUG
F000	
FFFF	ROM
E800	
E7FF	10K Microsoft BASIC
C000	
BFFF	BFF0-BFFF Microtan-65 I/O
BC00	1K I/O parts
BBFF	40K RAM TANRAM
2000	
1FFF	7K RAM onboard TANEX
0400	
03FF	1K RAM onboard Microtan-65
0000	

Figure 2: Memory map of the complete Microtan-65 system.





MPL is a simple yet powerful programming language, combining features from Pascal, BCPL, Fortran, and Forth. It is unusual in that its compiler is written in a portable subset of Microsoft Basic, making it useable on Nascom 2, Pet, Tandy etc. The following two sections give a brief description of MPL and its implementation.

1. The MPL Language.

The layout of an MPL program is similar to Pascal — statements are ended by a semicolon, and a space or spaces separate items. A program consists of a main section, followed by an optional list of function definitions; variables declared in the main section (by GVAR and GARR) are global, ie-accessible by all functions (like Fortran common). Example —

```
GVAR X=0 LENGTH=4 BREADTH=8;
GVAR CHAR='A';
```

— note that variable names can be of any length, and that global variables must be initialised to integers or characters, each being held in 2's complement 16 bit form. Arrays can also be declared —

```
GARR LIST 10. MESSAGE 5 = 'E'R.;
```

— which declares an array LIST of 10 elements, all zero, and MESSAGE of 5 elements, the first two set to the letters E and R.

The expression.

Having declared all our global variables, we can do a few calculations BUT, to reduce the size of the compiler, expressions (and function calls) are written in Reversed Polish, similar to Forth. A few examples should give you the idea.

MPL	Basic/Pascal
AREA =	AREA =
LENGTH BREADTH *	LENGTH * BREADTH
X = X 1 +;	X = X + 1
A = B C D * +;	A = B + C * D

The operators available in MPL are —

Arithmetical; + - * /
 Logical; GT LT LE GE EQ NE (meanings as Fortran)
 Array subscript ? Address generation @
 Boolean; AND OR NOT (False is zero, True is non — zero)
 Byte selection ; LBYTE RBYTE
 I/O; INCH OUTCH

So, the equivalent to the Basic expression (A>3) AND (A<10) is

```
A 3 GT A 10 LT AND;
```

In RP notation, brackets are not needed and are used for comments, which may be inserted anywhere, eg.

```
X (OUR ANSWER) = A B *;
```

In general the right-hand side of an '=' consists of constants, variables, and functions calls, and is referred to as an expression.

Repetition.

— similar to Pascal, though the loop termination method is similar to Algol 68.

Pascal	MPL
WHILE X> 3 DO	WHILE X 3 GT DO
BEGIN	X = X 1-;
X = X - 1;	SUM = SUM TERM +
SUM SUM + TERM	EWHILE;
END;	

MPL does not use BEGIN and END; the 'compound statement' is shown by special keywords, such as EWHILE, EFOR, EIF.

MPL also has —
 REPEAT
 statements
 UNTIL expression;

and

```
FOR variable = expression TO expression
DO
  statements
EFOR;
```

Looping is thus similar to Pascal, but is superior with regard to abnormal loop termination. MPL has the BCPL — style LOOP and BREAK statements, which transfer control back to the start of the loop, and to the statement following the loop respectively.

Selection.

Predictably — IF THEN ELSE EIF; (the ELSE is not mandatory.)

Pascal	MPL
IF X = 3 THEN	IF X 3 EQ THEN
BEGIN	
statements 1	statements 1
END	ELSE
ELSE	statements 2
BEGIN	EIF;
statements 2	
END;	

Functions.

— one of the most powerful features of MPL. Briefly —

1. Arguments are passed by value (but addresses can be passed).

2. Local variables and arrays (declared by LVAR and LARR) are allocated on a stack, thus allowing recursion, if a clash between a local and global name occurs, the local one is taken.

3. A function call is syntactically similar to the Forth style. Example — a simple function to find the largest of its two arguments —

```
DEF MAX A B ;
  IF A B GT THEN RESULT A
  ELSE RESULT B
EIF;
END;
```

Here, to transmit a result back to the calling program, we use a statement having the general form —

RESULT expression;

If the function has no results, just use RETURN.

A call of MAX could be —
X = 7 3 MAX;

X = C D E + MAX; (Equiv. to X = MAX (C, D+E))

or even

X = 8 7 13 MAX MAX;

The classic factorial definition is:

```
DEF FACT N;
  IF N 0 EQ THEN RESULT 1
  ELSE RESULT N 1 - FACT N *
EIF;
END;
Finally, note that functions can be used to implement 'read — only' variables, eg —
DEF BUFFSIZE;
  RESULT 128;
END;
```

— with the result that

IF X BUFFSIZE EQ THEN ... EIF is obeyed correctly, but BUFFSIZE = 44; ie . an attempt to alter BUFFSIZE, is rejected by the compiler.

Other features of MPL.

- address manipulation / pointer use.
- the easy addition of your own machine-code operators e.g., for graphics etc.)
- yes. Labels and the GOTO .
- arrays as function parameters, allowing one to write string functions.

2. The Compiler.

A compiler, NOT an interpreter. Like many portable compilers, MPL produces an intermediate code (M-code) that is easy to convert to any machine-code, or, more likely, to interpret. The advantages of MPL over other portable compilers are —

1. The compiler is written in a subset of Microsoft Basic, and its listing is manageable i.e. it is feasible to type it in. (Around 36C lines.) A mini — Pascal in Basic is given in Byte (Sept 78 — Chung / Yuen) but it consists of 630 lines of North Star (not Microsoft) Basic. Unlike MPL, it will not run in 16K.

2. Given the availability of 16 — bit arithmetic routines, an M-code always consists of a 3-letter code, then a space, then a number.

Unlike many systems, M-code is not described in English or flowcharts — instead, an interpreter consisting of a 100 — statement Basic program is the m-code definition. This can be used, or manually converted into assembly language. (In fact, the compiler and interpreter will fit together in 16K, allowing 'load — and — go' execution of small programs.)

Drawbacks.

1. The compiler is slow, being written in Basic.
2. Error diagnostics are primitive — the compiler stops when it detects an error. (This reduces the compiler size.)
3. The M-code interpreter written in Basic is usable, but being itself an interpreted program, is slow.



NASCOM NOTES

BY THE MERSEYSIDE NASCOM USERS GROUP

Firstly may I thank the staff of the Software Gazette for allowing me a few pages to give all you lucky Nascom owners some food for thought. If I get the free add in first you are more likely to read it rather than let you find the article totally useless and give up reading half way through. Meetings of the group take place on the first Wednesday of every month at the Mona Hotel in James Street Liverpool. We do not charge anything for visitors although you are well advised to bring along your cheque book to purchase some of the numerous goodies that we normally have to offer Nascom users including our new 8k/16k Eprom board (more about later) but be prepared to stand because space is becoming less and less available.

A large group of us visited the recent show in London but we were heartbroken to find that only one Nascom was there and that took some finding. It seems such a pity that the number of Nascom dealers is dwindling rapidly mainly it seems because of a lack of confidence in Nascom. I can only hope that the new Nascom management (?) can accomplish the impossible of restoring in dealers the faith and confidence that Nascom owners throughout the country have in the product. After all it was the first and most successful British machine produced and from its users have come a lot of engineers and technicians that are now employed in the trade.

Many months ago I decided to upgrade my Nascom 1 keyboard to the Nascom 2 spec. Trying to remember all the shift codes made me feel like shifting the whole machine out of the window. After many interesting hours the job was complete and the results were excellent. Comments from other users consisted of 'how can I do it?'. As usual I promised the earth but flesh being weak I could not persuade myself to document the changes required. Help arrived at last from G. Gibson of Manchester who had likewise carried the mods out but he had also written them down and he was kindly allowed me to print them for other users. If you do decide to carry out the changes ensure that you get the correct orientation of the keys. Refer to the diagram of the UNDERSIDE view otherwise when

you switch on you will get nothing out of the keyboard. Regarding the keys and availability I'm sorry but I can't help you. You could try Nascom I suppose.

Modifications to be made to the Nascom 1 keyboard.

Components required extra.

1. 1 * 22ohm resistor.
2. 1 * 2.2k resistor.
3. 1 * 1k resistor.

Link Ic3 pin 8 to Ic3 pin 12.

Link Ic3 pin 6 to Ic3 pin 13.

Link Ic3 pin 10 to Ic3 pin 11 to KSKT pin 7.

Link Ic3 pin 9 to Ic1 pin 11.

Solder 22ohm from common of inner resistor near Ic2 to Ic1 pin 10.

Solder 2k2 from common of outer resistors Ic2 to Ic1 pin 11.

Solder 1k from Graphics key B to Ic1 pin 9.

This next list is of all the new links that have to be made. I suggest that you tick them off as you complete them.

Graphics key A to Left key B.

Up key A to Down key B.

Right key A to CH key B.

S.B.C. key A to S.B.O. key B (* see note)

Left key A to Up key B.

Down key A to Right key B.

CH key A to S.B.C. key B.

NL key D to CH key C.

S.B.O. key A to 4 key A.

CH key D to BS key C.

- key D to CTRL key C.

CTRL key D to new shift key key C.

New shift key D to Ic5 pin 1.

X key C to Up key D.

Up key C to F key D.

Z key C to Left key D.
 Left key C to D key D.
 K key C to Down key D.
 Down key C to M key D.
 L key C to Right key D.
 Right key C to , key D.
 Q key C to Graphics key D.
 Graphics key C to 3 key D.
 Ø key C to S.B.O. key D.
 S.B.O. key C to P key D.
 R key D to S.B.C. key C.
 S.B.C. key D to 4 key C.
 Z key A to new shift key B.
 new shift key A to S key B.
 W key B to Ctrl key A.
 Ctrl key B to E key A.

* note S.B.C. = square brackets close.
 S.B.O. = square brackets open.

This next section is the most difficult to achieve and caution is needed! It involves cutting the track at various places and

requires a sharp knife. Also the holes for the new keys must be made and it has been found on the later boards that the holes coincide with tracks on the board so further alterations must be made to ensure continuity of the tracks.

Cut track from X key C.
 Cut track from Z key C.
 Cut track from K key C.
 Cut track from L key C.
 Cut track from Q key C.
 Cut track from Ø key C.
 Cut track from R key D.
 Cut track from Z key A.
 Cut track from W key B.
 Cut track from NL key D.
 Cut track from - key D.

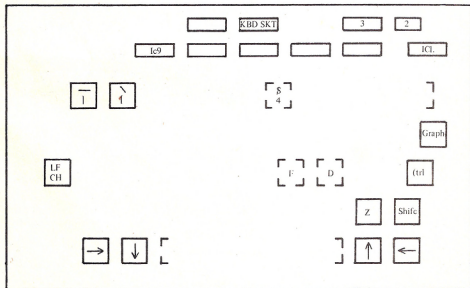
As you can see from the above list a lot of work is required to complete the task and if you lack confidence at all you do not attempt these mods but find a competent engineer to do the job for you.

Note there is no need to put all the new keys on the board but I'm afraid you will have to work out a shortened version that suits your requirements.



Underside view of
'Licon' keyswitch

Underside View of Nascom keyboard, showing position of new keys and keys mentioned in text.



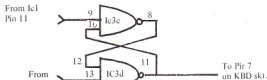
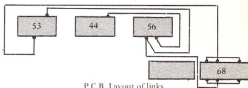


Diagram showing position of new keys within KBD matrix and wiring information around IC3 plus position of resistors marked *

Staying with hardware but concerning the Nascom 2. Because the machine was designed to be compatible with the various television standards throughout the world some liberties were taken with the frequencies. All 2 owners will know the problems associated with the frame speed i.e. displaced raster etc, but have you also noticed that on certain characters things are not quite right. For example the feet on the little man (bell character) are missing?

To get around this problem the boffins from the Merseyside group have come up with a solution (mainly because the chess characters we use on our graphics board took on funny looks).



P.C.B. Layout of links.

[1] Switch LSW1/6 should be set to 525 line operation (down)

MODIFICATIONS TO NASCOM 2 TO CORRECT VIDEO TIMING AND PROVIDE 16 LINES PER CHARACTER.

- ☐ LIFT IC53/11 & IC53/1
- ☐ LIFT IC56/5 & IC56/6
- ☐ LIFT IC68/1 & IC68/10
- ☐ LINK IC68/1 TO IC68/8 [0V] SET COUNT TO 13
- ☐ LINK IC68/10 TO IC68/16 [+SV]
- ☐ LINK IC53/11 TO IC56/5
- ☐ LINK IC56/5 TO IC68/5
- ☐ LINK IC56/6 TO IC44/11
- INVERT IN ROW SELECT
- ☐ LINK IC53/1 TO IC68/11 CLEAR ROW COUNT

Clear from prom now preloads 13 instead of 11 and sets top bit of row sel. Thus giving 19½ rows of 16 line chars. @ 65 US per line.

Frame frequency = 50.08HZ rather than 50.7HZ.

The display should now show 16 lines per char. As per nascom 1. The display may be a little higher than the original but much steadier.

[IC53/11 Indicates I.C. No 53 pin or leg No 11]

This mod disables the 625/525 line switch (LSW1/6) and must be set to 525 OP.

Lastly this month a basic program that I and many others have found to be most frustrating and well worth the effort of typing in but please note that it is written for Crystal basic and it will require modification to run on Microsoft basic. My thanks to Messrs Purcell and Hope for a very enjoyable program.

YOU WERE WALKING THROUGH THE WOODS,
AND YOU CAME ACROSS THE ENTRANCE OF A CAVE,
COVERED WITH BRUSH.

PEOPLE SAY THAT MANY YEARS AGO A PIRATE
HID HIS TREASURE IN THESE WOODS, BUT NOBODY
HAS EVER FOUND IT. IT MAY STILL BE HERE, YOU
NEVER KNOW!

)E1002

Quest - To Run On Nascom with X-TAL Basic Interpreter.

```

100 PRINTCHR$(30):FORJ=1TO35:POKEJ+3022,ASC(MID$("QUEST ADAPTED BY S.HOPE & M.PU
RCELL",J,1))
120 NEXT:PRINT"YOU WERE WALKING THROUGH THE WOODS."
150 PRINT"AND YOU CAME ACROSS THE ENTRANCE OF A CAVE."
160 PRINT"COVERED WITH BRUSH.":PRINT:PRINT"PEOPLE SAY THAT MANY YEARS AGO A PIRA
TE"
190 PRINT"HID HIS TREASURE IN THESE WOODS,BUT NOBODY"
200 PRINT"HAS EVER FOUND IT.IT MAY STILL BE HERE,YOU":PRINT"NEVER KNOW!"
400 READM9,T1,T2:DIMT(2),W(M9),M(S,M9),T$(3):FORJ=1TO3:READT$(J):NEXT
570 FORI=1TOM9:FORJ=0TO5:READM(J,I):NEXTJ,I:PRINT
905 PRINT"WHEN YOU ANSWER A QUESTION,I LOOK AT ONLY THE","FIRST LETTER,":GOSUB75
00
1000 D=0:T(1)=43:T(2)=0:N=S:M0=0:T(0)=T1:P=0:P1=0:FORJ=1TOM9:W(J)=0:NEXT
1100 PRINT:GOSUB 8000
1420 M0=M0+1:GOSUB6000:GOSUB4000
1500 IFT(0)=0ORN(1)STHEN1420
1700 GOSUB3000:PRINT:PRINT"CONGRATULATIONS:YOU GOT THE ";T$(1),"OUT IN ";
1750 PRINTM0:"MOVES AND YOU GOT";S+10:"POINTS"
1760 INPUT"WANT TO HUNT AGAIN?";A$:IFLEFT$(A$,1)=""V"STHEN6030
1790 STOP
3000 PRINTSIZE:S=0:FORJ=0TO2:IFT(J)=-1THENS=S+10
3010 NEXT:IFT(0)=-1THENS=S+10
3040 IFP=1THENS=S+10
3050 FORJ=2TOM9:S=S+W(J):NEXT:RETURN
4000 IFN=T2ORP=1ORT1=T2ORT(0)=-1THENRETURN
4070 IFN=1STHENP=160
4090 IFP1=0ORN=3STHENP1=P1+1
4120 IFP1=1STHENRETURN
4130 PRINT:PRINT"SUDDENLY THE PIRATE LEAPS OUT OF THE GLOOM","AND GRABS THE ";T$(
1):" "
4160 PRINT""HA!",HE SHOUTS,"YOU FOUND MY ";T$(1),"DID YOU:"
4170 PRINT"WELL,I'LL HIDE IT BETTER THIS TIME!","AS HE DISAPPEARS INTO THE DARK
NESS"
4210 P=1:T(0)=T2:RETURN
5000 A2=0:0$="" :INPUT0$:IFLEN(0$)>3THENS005
5002 FORJ=2TOLEN(0$)-1:IFMID$(0$,J,1)="" THEN0$=MID$(0$,J+1,1)
5003 NEXT:FORA2=1TO4:IF0$=MID$(0$,A2,1)THENS005
5004 NEXT:A2=0
5005 0$=LEFT$(0$,1):FORA1=1TO10:IF0$=MID$(0$,A1,1)THENRETURN
5010 NEXT:A1=0:RETURN
6000 N9=N:N8=0:GOSUB7000:IFN(1)THENN0=N:A0=A1
6120 PRINT:I=M(A1-1,N)
6200 IFI=-2THENI=N9
6220 IFI)500THENI=I-500:FORJ=0TO999:NEXT:GOTO6200
6300 ONI/100GOTO6340,6370
6320 N=I:GOTO6400
6340 N=I-100:IFT(0)=-1THENN=N+1
6350 GOTO6400
6370 N=I-200:IFT(0)=-1THENN=N+P
6400 IFN=1THENFORJ=0TO5:M(J,N)=2:NEXT:M(6-A0,N)=N0
6500 IFN8(2)THENGOSUB8000
6530 W(N)=1:N8=N
6600 IFM(0,N)=-2THENRETURN
6690 I=M(S,N):IFM(3,N)=100+AND(1)THENI=M(4,N)
6700 IFM(1,N)=100+AND(1)THENI=M(2,N)
6730 GOTO6200
7000 PRINT:PRINT"WHAT NEXT":GOSUB5000
7150 IFA1=0THENPRINT"PARDON?":GOSUB7500:GOSUB8000:GOTO7000
7300 IFA1(7)THENRETURN
7310 ONA1-6GOSUB7320,7330,7370,7430:GOTO7000
7320 GOSUB3000:PRINT"YOU HAVE":S:"POINTS":RETURN
7330 IFA2=0THENINPUT"TAKE WHAT";0$:0$="T"+0$:GOTO7499
7340 IFT(A2-1)=-1THENPRINT"YOU ALREADY HAVE THE ";T$(A2):RETURN
7341 IFT(A2-1)=NTHENT(A2-1)=-1:PRINT"OK,GOT THE ";T$(A2):RETURN
7342 PRINT"THE ";T$(A2):IFA2=1THENPRINT" ISN'T HERE":RETURN

```

```

7348 PRINT" AREN'T HERE":RETURN
7370 IFA2=0THENINPUT"LEAVE WHAT":D#Q#="L "+D#GOT07499
7380 IFT(A2-1)=-1THEN(TA2-1)=N:PRINT"OK.":RETURN
7383 PRINT"YOU DON'T HAVE THE ":T#(A2):RETURN
7430 IFA2=0THENINPUT"OPEN WHAT":D#Q#="O "+D#GOT07499
7440 IFN()GORA2()3THEN7465
7445 IFT(1)()-1THENPRINT"IT'S LOCKED.":RETURN
7450 PRINT"HMM.. FIRST KEY DOESN'T FIT.":FORJ=0TO999:NEXT
7455 PRINT"THE CUPBOARD IS WRENCHED OPEN...":IFT(2)()0THENPRINT"& IT'S EMPTY!!":
RETURN
7460 PRINT"& A BAG FALLS OUT,SPILLING OPEN ON THE FLOOR!":T(2)=N:GOTO8400
7465 IFN()44ORA2()4THEN7480
7468 IFD=1THENPRINT"IT'S ALREADY OPEN!":GOT07495
7470 IFT(1)()-1THEN7445
7475 D=1:M(4,44)=45:M(1,45)=44:PRINT"THE DOOR OPENS WITH A PROTESTING SCREECH":R
ETURN
7480 IFN=45ANDA2=4THENPRINT"CAN'T FIND A KEYHOLE":RETURN
7495 PRINT"YOU SURE GET WIERDOS PLAYING WITH COMPUTERS!":RETURN
7499 POP:GOSUB5002:GOT07150
7500 PRINT"THE COMMANDS ARE: NORTH, SOUTH, EAST, WEST, UP, DOWN, TAKE, LEAVE":
7510 PRINT", POINTS AND OPEN.":RETURN
8000 I=INT(N/5):J=N-5*I+1:PRINT"YOU'RE ":
8100 ON1+IGOT08200,8210,8220,8230,8240,8250,8260,8270,8280,8290
8200 ONJGOSUB9000,9010,9020,9030,9040:GOT08400
8210 ONJGOSUB9050,9060,9070,9080,9090:GOT08400
8220 ONJGOSUB9100,9110,9120,9130,9140:GOT08400
8230 ONJGOSUB9150,9160,9170,9180,9190:GOT08400
8240 ONJGOSUB9200,9210,9220,9230,9240:GOT08400
8250 ONJGOSUB9250,9260,9270,9280,9290:GOT08400
8260 ONJGOSUB9300,9310,9320,9330,9340:GOT08400
8270 ONJGOSUB9350,9360,9370,9380,9390:GOT08400
8280 ONJGOSUB9400,9410,9420,9430,9440:GOT08400
8290 ONJGOSUB9450,9460,9470,9470,9470:GOT08400
8300 IFJ=1THENPRINT" IS HERE!":RETURN
8310 PRINT" ARE HERE!":RETURN
8400 FORJ=1TO3:IFT(J-1)=NTHENPRINT:PRINT"THE ":T#(J):GOSUB8300
8500 NEXT:IFT(0)()T2ORT1=T2ORT1()NTHENRETURN
8530 PRINT:PRINT"A NOTE ON THE WALL SAYS 'PIRATES NEVER LEAVE'"
8560 PRINT"THEIR TREASURE TWICE IN THE SAME PLACE!":RETURN
9000 DATA9,45,12,"TREASURE CHEST","KEYS","GOLD COINS"
9010 DATA0,0,0,0,0,0
9011 PRINT"AT A DEAD END!!":RETURN
9020 DATA-2,101,-2,0,0,0:PRINTCHR$(28);"YOU CAN'T GO IN THAT DIRECTION":RETURN
9030 DATA33,2,1,10,106,4:PRINTCHR$(28);"A TUNNEL GOES NORTH-SOUTH.THERE IS AN OP
ENING"
9031 PRINT"TO THE WEST.":RETURN
9040 DATA3,30,2,11,2,1:PRINT"ON THE BRINK OF A PIT.":RETURN
9050 DATA8,8,15,10,8,16:PRINT"OUTSIDE THE CAVE.GO SOUTH TO ENTER.":RETURN
9060 DATA16,3,2,10,2,2:PRINT"AT THE HOME F THE GNOME-KING.LUCKILY,","HE'S OUT FO
R THE DAY."
9061 PRINT"HERE IS A LARGE,WHITE CUPBOARD ON THE WALL.":RETURN
9070 DATA-2,101,-2,0,0,0:PRINTCHR$(28);"THE GNOME-KING IS HERE! YOU'D BETTER GET
OUT!":RETURN
9080 DATA18,18,15,10,18,9:PRINT"LOST IN THE WOODS.":RETURN
9090 DATA-2,45,5,1,0,-2
9091 PRINTCHR$(28):RETURN
9100 DATA-2,101,-2,0,0,0:PRINT"NOT GOING TO GET FAR,DIGGING THRU ROCK":RETURN
9110 DATA1,13,4,2,1,2:PRINT"AT THE BOTTOM OF A PIT."
9111 PRINT"A LITTLE STREAM FLOWS OVER THE ROCKS HERE.":RETURN
9120 DATA36,2,1,2,1,2:GOT09011
9130 DATA2,37,2,1,11,14:PRINT"AT A WIDE SPOT.THERE IS A SOOTY PATCH"
9131 PRINT"WHERE SOMEBODY HAS RESTED A TORCH AGAINST THE"
9132 PRINT"WALL.THERE ARE JAGGED ROCKS ABOVE YOU.":RETURN
9140 DATA13,1,19,2,31,31:PRINT"IN A CANYON,HIGH ON THE WALL ABOVE YOU IS":
9141 PRINT"SCRATCHED THE MESSAGE 'BILBO WAS HERE'":RETURN
9150 DATA-2,101,-2,0,0,0:PRINT"NOT A BIRD. YOU CAN'T FLY!":RETURN
9160 DATA5,33,2,10,1,106:PRINT"IN A LOW CHAMBER,A TIGHT TUNNEL GOES"

```

```

9161 PRINT"EAST,AND YOU CAN WALK SOUTH OR WEST.THERE IS","LIGHT TO THE NORTH":RE
TURN
9170 DATA-2,101,-2,0,0,0:PRINTCHR$(28);"IT'S A TIGHT SQUEEZE.YOU CAN'T GET PAST
WITH THE":
9171 PRINT$(1):RETURN
9180 DATA-2,101,8,0,0,0:PRINTCHR$(28);"I DON'T THINK YOU CAN FIND THE CAVE.":RET
URN
9190 DATA224,2,2,14,1,42:PRINT"AT THE TOP OF A CLIMB.BELOW YOU"
9191 PRINT"A MESSAGE SAYS 'BILBO WAS HERE'.":RETURN
9200 DATA226,1,2,2,25,45:PRINT"AT THE NORTH SIDE OF A CHASM,TOO WIDE TO"
9201 PRINT"JUMP.RINGING ECHOES FROM BELOW ARE THE ONLY"
9202 PRINT"INDICATION OF DEPTH.A ROTTING ROPE SUSPENSION","BRIDGE SPANS THE CHAS
M.":RETURN
9210 DATA1,226,2,46,39,25:PRINT"IN XANADU.BELOW YOU ALPH,THE SACRED"
9211 PRINT"RIVER RUNS THROUGH CAVERNS MEASURELESS TO MAN","DOWN TO A SUNLESS SEA
.":RETURN
9220 DATA-2,33,13,50,29,30:GOTO9091
9230 DATA2,1,2,31,2,44:PRINT"ON THE LEDGE ABOVE THE GUILLOTINE ROOM.":RETURN
9240 DATA-2,101,19,0,0,0:PRINTCHR$(28);"I HEAR THE GIANT THERE! YOU'D BETTER GO
BACK.":RETURN
9250 DATA21,20,2,2,1,19:PRINT"IN THE GIANT'S CAVERN.BETTER LEAVE BEFOREHE COMES!
":RETURN
9260 DATA-2,65,-2,50,11,14:PRINT"IN THE QUEST RESEARCH & DEVELOPMENT AREA"
9261 PRINT"SORRY,NO VISITORS ALLOWED.YOU'LL HAVE TO LEAVE.":RETURN
9270 DATA2,40,2,2,21,20:PRINT"IN THE CRYSTAL PALACE.THE WALLS"
9271 PRINT"RESONATE WITH AWESOME MUSIC."
9280 DATA-2,60,221,50,14,19:GOTO9091
9290 DATA2,42,2,13,1,1:PRINT"AT THE TOP OF A GIANT STALACTITE.YOU"
9291 PRINT"COULD SLIDE DOWN,BUT YOU COULDN'T GET BACK UP.":RETURN
9300 DATA34,34,2,1,4,43:PRINT"IN A LITTLE GROTTO.THERE IS A BOOK HERE."
9301 PRINT"JANE'S FIGHTING SHIPS (1763)":RETURN
9310 DATA14,14,23,2,1,2:PRINT"IN THE GUILLOTINE ROOM.A SHARP ROCK"
9311 PRINT"BALANCES PRECARIOUSLY ON THE LEDGE ABOVE YOU.":RETURN
9320 DATA-2,101,516,0,0,0:PRINT"IN A SHUTE,SCRAMBLING DOWN THE ROCKS."
9321 PRINT"NO WAY TO STOP! HANG ON!":RETURN
9330 DATA2,1,2,1,116,3:PRINTCHR$(28);"THE TIGHT TUNNEL TURNS A CORNER.":RETURN
9340 DATA1,35,2,1,30,30
9341 PRINT"IN A LITTLE,TWISTY MAZE.":RETURN
9350 DATA2,1,2,37,34,36:GOTO9341
9360 DATA35,2,1,37,34,12:GOTO9341
9370 DATA2,1,35,2,13,2:GOTO9341
9380 DATA2,21,2,116,1,2:PRINT"IN A PREHISTORIC DWELLING.ON THE WALLS"
9381 PRINT"ARE DRAWINGS OF BISON DONE IN RED CLAY.THE FLOORIS STREWN WITH BONES.
":
9382 PRINT"THE REMAINS OF ANCIENT","RITUALS.A SMALL TUNNEL GOES THROUGH THE FLOO
R.":RETURN
9390 DATA2,40,2,32,21,26:PRINT"IN A BLACK HOLE."
9391 PRINT"THE FORCE OF GRAVITY IS OVERWHELMING!":RETURN
9400 DATA40,40,2,2,40,41:PRINT"IN THE LABYRINTH.":RETURN
9410 DATA40,40,40,2,40,39:PRINT"IN A LABYRINTH.IT'S VERY DARK IN HERE!":RETURN
9420 DATA28,28,28,28,28,28:PRINT"IN THE ASHRAM.THE AIR IS HEAVY WITH"
9421 PRINT"INCENSE.ALL DIRECTIONS SEEM THE SAME...":RETURN
9430 DATA2,30,2,2,2,2:GOTO9011
9440 DATA23,1,2,2,2,2:PRINT"IN A WIDE,LOW CAVERN.LITTERED WITH ROCKS.":
9441 PRINT"THERE IS A SMALL,OAK DOOR ON THE WESTERN SIDE.":RETURN
9450 DATA20,2,2,2,2,2:PRINT"ON THE SOUTH SIDE OF A BOTTOMLESS CHASM."
9451 PRINT"THERE IS A DOOR TO THE EAST.":RETURN
9460 DATA2,47,21,2,48,2
9461 PRINT"BEHIDE THE RIVER ALPH.":RETURN
9470 DATA2,2,2,2,46,1
9471 GOSUB9461:PRINT"THE ROOF DIPS TO MEET THE WATER":RETURN
9480 DATA2,46,2,2,2,2:GOTO9471
Ok

```

Footnote.

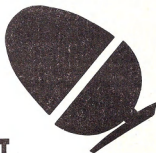
Anybody wishing to purchase an Eprom/ROM board send cheque for £40 plus V.A.T. to Microdigital for information on this board write to the Treasurer Merseyside Nascom User

Group c/o. T. Lyon and Co. Ltd., Samuel House, Taylor St. Liverpool. S.A.E. please. Graham N. Myers. Secretary Merseyside Nascom User Group.

A number processor on the ACORN

LAURENCE
HARDWICK

PART II



Well here we are after a short summer break with the second part of this article. The first instalment in the May issue described the hardware required to interface the National semiconductors MM57109 Number processor with an Acorn system one. This part describes a program which may be loaded into an Acorn system one with this interface, and turns the system into a programmable calculator.

Loading the program

The program given in listing 1 may be typed into the Acorn in the usual manner and saved on tape. The program must be saved in two sections one section is between locations 0300 and 03FE and the other between 0EB0 and 0EB4. The start point of the program is at label AAAAA which is at location 0358 in memory.

Using the editor

When the program is executed the first section is an editor, this editor is used to enter a program for the number processor. The display will show:—

```
00 XX
```

Where XX is some random two digit number. The editor is now ready for you to enter a program for the number processor, the key layout is as shown in figure 1. As a key is pressed the code to be sent to the number processor is stored in a buffer, and the buffer pointer, which is shown by the 1st two digits on the display, is incremented. In this way a program for the processor is built up in the buffer.

As the processor understands many more instructions than there are keys on the Acorn some keys have several functions. The m.g and p keys on the acorn are used to select the alternative key functions (Fn 1,2 or 3) of each key. For example to enter the constant PI you should type "Fn 2" (The g key) and then "8".

Some number processor instructions require two words, the 1st word is the code for the instruction, and the 2nd is an address. In the case of a JUMP instruction the address is the location in the program buffer that execution should pass to if the jump is made. When a JUMP instruction is entered the editor recognises this

and is prepared to receive an address. The address should be entered using the keys 0–f, any mistakes can be altered by entering the correct number. When the address is correct the key should be pressed and program entry continued in the normal way. In the case of the OUT instruction the address is the location in page zero that will be used to store the answer that is output by the number processor. When an OUT instruction is entered the editor recognises this and automatically sets the 2nd instruction word to 10, the number when output will be stored in a buffer at 0010. The number when stored at this location is ready for display by the Acorn monitor display routine. However results may be stored anywhere in page zero, and the address may be entered in the same way as for the JUMP instruction. When the address is correct or if it is not to be changed from 10 the key should be pressed. Program entry then continues in the normal way. Most programs will end with a HALT instruction (code "Fn 2" ".") as a signal that the number processor should cease execution of the program and control will return to the editor section. If a program does not end with a HALT instruction the number processor may continue on through the buffer executing random instruction and producing wierd results. When a program has been entered the "4" and "V" keys may be used to step up and down through the program inspecting it in much the same way as the Acorn monitor. Any mistakes in the program may be corrected at this stage. When inspecting a program the second two digits in the display are the code to be sent to the number processor. Reference may be made to the table of instruction codes in the first part of this article to check that the correct codes are in the buffer.

Running a program

When the r key is pressed the instructions stored in the program buffer are sent to the number processor. The instructions are sent one at a time starting at location 00. Between each instruction send the Acorn monitors display routine is called to display the contents of the

display buffer at location 0010. The contents of this buffer will be changed as a program is running by any OUT instruction that stores output in this buffer. In this way the progress of a program may be noted if it contains OUT instructions at various stages through the calculation. When a HALT instruction encountered processing stops and the current contents of the display buffer are shown, any key may then be pressed to return to the editor.

At any time during the running of a program execution may be halted and the current display held by pressing any key (Other than the S key). Execution will continue when the key is released. If the s key is pressed while a program is running execution will halt and the current contents of the display register will be shown, any key may then be pressed to return to the editor.

If any errors occur during program execution the message "Error" will be displayed and again any key may be pressed to return to the monitor.

The program may then be corrected or changed using the editor and restarted.

Tape interface

Once a program has been entered into the buffer it may be saved on tape by using the s key. The tape recorder should be connected in the usual way and switched to record. The start and end addresses of the data to be saved are automatically calculated and transmission is started as soon as the s key is pressed.

Programs may be loaded off tape in the usual way by switching the tape recorder to play and pressing the l key.

Example program

The listing below is for a simple program to load 0.005 into the X register of the number processor and then repeatedly multiply it by 9 displaying the result of each multiplication. The program will stop when the result is greater than 1×10^9 and an overflow error occurs. The program demonstrates the way that display switches automatically from floating point to scientific notation. The use of a key to hold (or the s key to halt) execution of the program may also be demonstrated. The number processor works in Reverse Polish and the program also demonstrates the use of this system.

DISPLAY	INSTRUCTION	ACORN	KEYS	BUFFER	ENTRY
00 XX		0		00	
01 XX		.	A	0A	
02 XX		0	0	00	
03 XX		0	0	00	
04 XX		5	5	05	
05 XX	ENTER	g, 1		21	
06 XX		9	9	09	
07 XX	*	E		3b	
08 XX	OUT(10)	m, 6, ↑		1610	
0A XX	JMP(06)	m, 5, 0, 6		1506	

Figure 1 KEY LAYOUT

X → M RAD → DEG	M → X DEG → RAD	Xm ← POP	Xm → CLR NOP	Fn 1 Fn 2 Fn 3
+	-	*	M ↔ X	
PI Y ↑ X 8	IBN2	DBN2 HALT		
J, E=1 SIN SQRT 4	JMP COS Ln X 5	OUT TAN LOG X 6	+/- 1/X 7	
INV X ↔ Y 0	J, X=0 ENTER EXP 1	J, X<0 10↑X 2	J, X <1 ROL X↑2 3	

Fn 1	Load
Fn 2	Run
Fn 3	↑
save	V

NCI ACORN 6502 Assembler Page 01

```

0010: 0200      NCI      ORG      $0200      file 6
0020:
0030:
0040:
0050:
0060:
0070:
0080:
0090:

```

 *
 * CALCULATOR *
 * TO RUN ON *
 * ACORN SYSTEM *
 * ONE WITH *
 * MM57109 *

0100:	0200	FROM	*	\$0006	Acorn save and load pointers
0110:	0200	TO	*	FROM	+02
0120:	0200	KEY	*	\$000D	Last key pressed
0130:	0200	DISREG	*	\$0010	Display register
0140:	0200	REGPNT	*	\$0020	Pointer for number output
0150:	0200	TEMPX	*	REGPNT	+02
0160:	0200	POINT	*	TEMPX	+01 Input instruction pointer
0170:	0200	TABLE	*	\$FFEA	Font for digits
0180:	0200	DISPLY	*	\$FE0C	Display scan routine
0190:	0200	DHEXTD	*	\$FE6F	Display accumulator
0200:	0200	MEMDIS	*	\$FE5E	Display data
0210:	0200	COMINC	*	\$FEA0	Use for save and load
0220:	0200	GETBYT	*	\$FEDD	Byte from tape
0230:	0200	PUTBYT	*	\$FEB1	Byte to tape
0240:	0200	STACK	*	\$0980	
0250:	0200	PIA	*	\$0900	Second 8154 on Acorn
0260:	0200	DRA	*	PIA	+20
0270:	0200	DRB	*	PIA	+21
0280:	0200	DDRA	*	PIA	+22
0290:	0200	DDRB	*	PIA	+23
0300:	0200	TESTA	*	PIA	
0310:	0200	SETB	*	PIA	+18
0320:	0200	CLEARB	*	PIA	+08
0330:	0200 A9 FF	CALC	LDAIM	\$FF	Set up PIA and send reset
0340:	0202 8D 23 09		STA	DDRB	
0350:	0205 A9 1F		LDAIM	\$1F	Set single scan of display
0360:	0207 85 0E		STA	\$000E	
0370:	0209 A9 40		LDAIM	\$40	
0380:	020B 8D 21 09		STA	DRB	
0390:	020E A2 0A		LDXIM	\$0A	
0400:	0210 CA	WAIT	DEX		Delay for length of reset pulse.
0410:	0211 D0 FD		BNE	WAIT	
0420:	0213 8D 1F 09		STA	SETB	+07 Clear reset
0430:	0216 AD 06 09	EWRITE	LDA	TESTA	+06
0440:	0219 30 FB		BMI	EWRITE	Wait for MM57109 to settle
0450:	021B A9 2F		LDAIM	\$2F	Master reset
0460:	021D 20 29 03		JSR	PUTIT	
0470:	0220 A9 22		LDAIM	\$22	Toggle to scientific mode
0480:	0222 20 29 03		JSR	PUTIT	
0490:	0225 20 0B 03	TEST	JSR	NEXT	Send next instruction
0500:	0228 C9 0F		CMPIM	\$0F	End of program marker
0510:	022A F0 17		BEQ	RUNOUT	
0520:	022C C9 16	NOTEND	CMPIM	\$16	Output number instruction
0530:	022E F0 3C		BEQ	OUTPUT	Special treatment for output
0540:	0230 C9 1B		CMPIM	\$1B	Check for jump instruction
0550:	0232 B0 04		BCS	RDYEXT	Not jump so scan display
0560:	0234 C9 10		CMPIM	\$10	NCI
0570:	0236 B0 0C		BCS	JUMP	Special treatment for jumps
0580:	0238 20 0C FE	RDYEXT	JSR	DISPLY	Scan display
0590:	023B A5 0F		LDA	\$000F	Check for key pressed
0600:	023D 30 E6		BMI	TEST	Send next instruction when clear
0610:	023F C9 2B		CMPIM	\$2B	Check for stop key
0620:	0241 D0 F5		BNE	RDYEXT	
0630:	0243 60		RUNOUT	RTS	
0640:	0244 20 0B 03	JUMP	JSR	NEXT	Just because the 57109 expects it
0650:	0247 2C 05 09	JMPLOP	BIT	TESTA	+05
0660:	024A 30 07		BMI	DOJUMP	The jump is to be made
0670:	024C 2C 07 09		BIT	TESTA	+07 Test for ready
0680:	024F 30 D4		BMI	TEST	And send next instruction
0690:	0251 10 F4		BPL	JMPLOP	
0700:	0253 AA	DOJUMP	TAX		Load new pointer

0710:	0254	4C	25	02		JMP	TEST	and continue
0720:	0257	2A			AGAIN	ROLA		This routine accepts
0730:	0258	10	0C			BPL	ACCEPT	a stream of output from
0740:	025A	6A				RORA		the number processor and stores
0750:	025B	29	0F			ANDIM	\$0F	it in page zero indexed by Y
0760:	025D	99	00	00		STAA	\$0000	
0770:	0260	C8				INY		
0780:	0261	2C	06	09	ERW	BIT	TESTA	+06
0790:	0264	30	FB			BMI	ERW	
0800:	0266	AD	20	09	ACCEPT	LDA	DRA	Entry point of routine
0810:	0269	10	EC			BPL	AGAIN	
0820:	026B	60				RTS		
0830:	026C	20	0B	03	OUTPUT	JSR	NEXT	Get a number out
0840:	026F	85	20			STA	REGPNT	Program points to store in
0850:	0271	A9	00			LDAIM	\$00	page zero somewhere
0860:	0273	85	21			STA	REGPNT	+01
0870:	0275	A0	30			LDYIM	\$30	Pointer for accept routine
0880:	0277	20	66	02		JSR	ACCEPT	and get output
0890:	027A	A5	30			LDA	\$0030	Check exponent
0900:	027C	D0	0C			BNE	DUNOUT	if it is greater than six
0910:	027E	A5	31			LDA	\$0031	we print out in scientific form
0920:	0280	C9	06			CMPI	\$06	
0930:	0282	B0	06			BCS	DUNOUT	
0940:	0284	20	90	02		JSR	FLOAT	otherwise we can float it
0950:	0287	4C	25	02		JMP	TEST	
0960:	028A	20	C9	02	DUNOUT	JSR	SCIENC	Scientific display
0970:	028D	4C	25	02		JMP	TEST	
0980:	0290	A9	22		FLOAT	LDAIM	\$22	Toggle mode to floating point
0990:	0292	20	29	03		JSR	PUTIT	
1000:	0295	A9	16			LDAIM	\$16	Ask for output again
1010:	0297	20	29	03		JSR	PUTIT	
1020:	029A	20	29	03		JSR	PUTIT	dummy put
1030:	029D	A0	30			LDYIM	\$30	
1040:	029F	20	66	02		JSR	ACCEPT	We now get floating point output
1050:	02A2	A9	22			LDAIM	\$22	
1060:	02A4	20	29	03		JSR	PUTIT	toggle back
1070:	02A7	38			PRINT	SEC		And create floating point display
1080:	02A8	A5	31			LDA	\$0031	Adjust decimal point position
1090:	02AA	E9	04			SBCIM	\$04	
1100:	02AC	85	31			STA	\$0031	
1110:	02AE	A0	07			LDYIM	\$07	
1120:	02B0	B9	31	00	MORE	LDAAY	\$0031	Get next number NCI
1130:	02B3	20	00	03		JSR	HEX	Translate into font
1140:	02B6	C6	31			DEC	\$0031	Check for decimal point
1150:	02B8	D0	02			BNE	NOPOIN	
1160:	02BA	09	80			ORAIM	\$80	Add decimal point
1170:	02BC	91	20		NOPOIN	STAIY	REGPNT	And store in display register
1180:	02BE	88				DEY		
1190:	02BF	D0	EF			BNE	MORE	Go round again
1200:	02C1	A5	30			LDA	\$0030	Load sign
1210:	02C3	0A				ASLA		Adjust to segment \$40
1220:	02C4	0A				ASLA		
1230:	02C5	0A				ASLA		
1240:	02C6	91	20			STAIY	REGPNT	And store in display register
1250:	02C8	60				RTS		
1260:	02C9	A0	07		SCIENC	LDYIM	\$07	Format scientific display
1270:	02CB	A5	31			LDA	\$0031	Load exponent
1280:	02CD	20	00	03		JSR	HEX	Translate to font
1290:	02D0	91	20			STAIY	REGPNT	and store in display register
1300:	02D2	A5	30			LDA	\$0030	Same for most significant

1310:	02D4	20	00	03		JSR	HEX	exponent digit
1320:	02D7	88				DEY		
1330:	02D8	91	20			STAIY	REGPNT	
1340:	02DA	A5	32			LDA	\$0032	Load sign of exponent
1350:	02DC	6A				RORA		Adjust for correct segment
1360:	02DD	6A				RORA		
1370:	02DE	6A				RORA		
1380:	02DF	88				DEY		
1390:	02E0	48				PHA		Save sign of mantissa
1400:	02E1	29	40			ANDIM	\$40	Mask off sign
1410:	02E3	91	20			STAIY	REGPNT	Display sign
1420:	02E5	88				DEY		
1430:	02E6	B9	33	00	SCILOP	LDAAY	\$0033	Now do mantissa
1440:	02E9	20	00	03		JSR	HEX	Get font for digit
1450:	02EC	91	20			STAIY	REGPNT	and display it
1460:	02EE	88				DEY		
1470:	02EF	D0	F5			BNE	SCILOP	For all of mantissa
1480:	02F1	C8				INY		
1490:	02F2	09	80			ORAIM	\$80	Add decimal point to last digit
1500:	02F4	91	20			STAIY	REGPNT	and display it
1510:	02F6	88				DEY		
1520:	02F7	68				PLA		Reload sign of mantissa
1530:	02F8	6A				RORA		Adjust to correct segment
1540:	02F9	6A				RORA		
1550:	02FA	6A				RORA		
1560:	02FB	29	40			ANDIM	\$40	
1570:	02FD	91	20			STAIY	REGPNT	and display it
1580:	02FF	60				RTS		
1590:	0300	86	22		HEX	STX	TEMPX	Convert number into correct
1600:	0302	29	0F			ANDIM	\$0F	font for Acorn display
1610:	0304	AA				TAX		
1620:	0305	BD	EA	FF		LDAAX	TABLE	Load font
1630:	0308	A6	22			LDX	TEMPX	and restore X
1640:	030A	60				RTS		
1650:	030B	BD	80	09	NEXT	LDAAX	STACK	Get next byte of instruction list
1660:	030E	E8				INX		Increment pointer to instructions
1670:	030F	20	29	03		JSR	PUTIT	
1680:	0312	2C	04	09		BIT	TESTA	+04 NCI
1690:	0315	10	26			BPL	PUTRET	
1700:	0317	A0	07		ERROR	LDYIM	\$07	Display error message
1710:	0319	B9	3E	03	ELOOP	LDAAY	EMESS	Load from message
1720:	031C	99	10	00		STAAY	DISREG	and store to display
1730:	031F	88				DEY		
1740:	0320	10	F7			BPL	ELOOP	For all eight characters
1741:	0322	68				PLA		
1742:	0323	68				PLA		
1750:	0324	A9	2B			LDAIM	\$2B	Send error flag clear
1760:	0326	20	29	03		JSR	PUTIT	
1800:	0329	2C	07	09	PUTIT	BIT	TESTA	+07 Send byte to MM57109
1810:	032C	10	FB			BPL	PUTIT	Wait for ready pulse
1820:	032E	48			ENDPUT	PHA		
1830:	032F	09	C0			ORAIM	\$C0	Set hold and reset bits of byte
1840:	0331	8D	21	09		STA	DRB	and send it
1850:	0334	2C	07	09	NEXLOP	BIT	TESTA	+07 Wait for ready to go low
1860:	0337	30	FB			BMI	NEXLOP	
1870:	0339	8D	0E	09		STA	CLEARB	+06 Then clear hold line
1880:	033C	68				PLA		
1890:	033D	60			PUTRET	RTS		
1900:	033E	00			EMESS	=	\$00	Table of error message font
1910:	033F	79				=	\$79	
1920:	0340	50				=	\$50	

1930:	0341	50	=	\$50	
1940:	0342	5C	=	\$5C	
1950:	0343	50	=	\$50	
1960:	0344	00	=	\$00	
1970:	0345	00	=	\$00	
ID 07					
0010:	0346	A2 23	UPDATE	LDXIM	POINT file 7
0020:	0348	20 5E FE		JSR	MEMDIS Display instruction
0030:	034B	38		SEC	
0040:	034C	A5 23		LDA	POINT And address of instruction
0050:	034E	E9 80		SBCIM	\$80 Adjusted for start of table
0060:	0350	A0 03		LDYIM	\$03 Displayed in third and fourth positions
0070:	0352	20 6F FE		JSR	DHEXTD
0080:	0355	4C 0C FE		JMP	DISPLY display until key pressed and return
0090:	0358	D8	AAAAA	CLD	Entry point start here
0100:	0359	A9 09		LDAIM	STACK / Set up zero page locations
0110:	035B	85 24		STA	POINT +01
0120:	035D	A9 80		LDAIM	STACK
0130:	035F	85 23		STA	POINT
0140:	0361	A9 00		LDAIM	\$00 And clear display
0150:	0363	85 10		STA	DISREG
0160:	0365	85 11		STA	DISREG +01
0170:	0367	85 12		STA	DISREG +02
0180:	0369	85 15		STA	DISREG +05
0190:	036B	20 46 03	ROUND	JSR	UPDATE Main loop display curret instruction
0200:	036E	B0 12		BCS	FUNK Branch if function key
0210:	0370	C9 0C		CMPI	\$0C Keys less than 0C are stored as keyed
0220:	0372	90 02		BCC	FOUND
0230:	0374	69 2C		ADCIM	\$2C Add offset for +/-*
0240:	0376	81 00	FOUND	STAIX	\$00 And save instruction
0250:	0378	E6 23	UP	INC	POINT Increment instruction pointer
0260:	037A	30 EF		BMI	ROUND And display again
0270:	037C	C6 23	DOWN	DEC	POINT Decrement insruction pointer
0280:	037E	30 EB		BMI	ROUND And display again NCI
0290:	0380	10 F6		BPL	UP Too far down so step up a bit
0300:	0382	29 07	FUNK	ANDIM	\$07 Function key so mask key
0310:	0384	C9 06		CMPI	\$06
0320:	0386	F0 F0		BEQ	UP 6 is up arrow
0330:	0388	B0 F2		BCS	DOWN 7 is down arrow
0340:	038A	C9 05		CMPI	\$05
0350:	038C	D0 0D		BNE	NORUN 5 is r for run
0360:	038E	20 00 02		JSR	CALC Do calculation
0370:	0391	A9 FF		LDAIM	\$FF reset display for multy scan
0380:	0393	85 0E		STA	\$000E
0390:	0395	20 0C FE		JSR	DISPLY This displays the answer until key
0400:	0398	4C 58 03		JMP	AAAAA is pressed and then restart
0420:	039B	C9 03	NORUN	CMPI	\$03
0430:	039D	90 03		BCC	SHIFT M,G and P keys are instruction shift
0440:	039F	4C DF 03		JMP	SAVL0D S and L keys for save and load
0450:	03A2	85 10	SHIFT	STA	DISREG Put key code on display
0460:	03A4	E6 10		INC	DISREG To give som indication of shift
0470:	03A6	20 46 03		JSR	UPDATE And get another key
0480:	03A9	B0 D7		BCS	FUNK Function kes are to be obeyed
0490:	03AB	A5 10		LDA	DISREG Get shift key back
0500:	03AD	84 10		STY	DISREG And clear display digit

0510: 03AF 0A	ASLA	Get bits into correct place
0520: 03B0 0A	ASLA	
0530: 03B1 0A	ASLA	
0540: 03B2 0A	ASLA	
0550: 03B3 05 0D	ORA KEY	And add in new key
0560: 03B5 C9 1B	CMPIM \$1B	Values below this are two word
0570: 03B7 90 0C	BCC UPPER	instructions so get upper word
0580: 03B9 C9 2B	CMPIM \$2B	Or transform other keys to correct code
0590: 03BB B0 B9	BCS FOUND	
0600: 03BD C9 27	CMPIM \$27	
0610: 03BF 90 B5	BCC FOUND	
0620: 03C1 E9 1B	SBCIM \$1B	
0630: 03C3 D0 B1	BNE FOUND	Branch always
0640: 03C5 81 00	UPPER STAIX \$00	Store instruction in stack
0650: 03C7 A9 10	LDAIM \$10	Load default option of 10
0660: 03C9 E6 23	INC POINT	And store as next byte
0670: 03CB 81 00	STAIX \$00	Then get new upper word if required
0680: 03CD 20 46 03	JUMING JSR UPDATE	Main loop display current word
0690: 03D0 B0 B0	BCS FUNK	Function key terminates update
0700: 03D2 A1 00	LDAIX \$00	Load old word
0710: 03D4 0A	ASLA	Shift bits up
0720: 03D5 0A	ASLA	
0730: 03D6 0A	ASLA	
0740: 03D7 0A	ASLA	
0750: 03D8 05 0D	ORA KEY	Add in new key
0760: 03DA 81 00	STAIX \$00	And save as new word
0770: 03DC 4C CD 03	JMP JUMING	And do until exit
0780: 03DF D0 03	SAVL0D BNE LOAD	L key for load
0790: 03E1 4C 80 0E	JMP SAVE	Or S for save routine in PIA ram
0800: 03E4 A2 04	LOAD LDXIM \$04	Load the same as in monitor
0810: 03E6 20 DD FE	ADRL0P JSR GETBYT	but must be copied here
0820: 03E9 95 05	STAIX \$05	In order to return
0830: 03EB CA	DEX	
0840: 03EC D0 F8	BNE ADRL0P	
0850: 03EE 20 DD FE	LODDAT JSR GETBYT	NCI
0860: 03F1 81 06	STAIX \$06	
0870: 03F3 8D 21 0E	STA \$0E21	
0880: 03F6 20 A0 FE	JSR COMINC	
0890: 03F9 D0 F3	BNE LODDAT	
0900: 03FB 4C 58 03	JMP AAAAA	Return to start when done
0910:		
ID 08		
0010: 0E80	SAVE ORG \$0E80	Save find the end of the
0020: 0E80 A9 09	LDAIM \$09	Instruction list automatically
0030: 0E82 85 07	STA FROM	+01 so the start and end of memory
0040: 0E84 85 09	STA TO	+01 to be saved are known
0050: 0E86 A0 00	LDYIM \$00	
0060: 0E88 84 23	STY POINT	
0070: 0E8A A0 80	LDYIM \$80	
0080: 0E8C 84 06	STY FROM	
0090: 0E8E B1 23	FEND LDAIX POINT	Find end of programm
0100: 0E90 C8	INY	
0110: 0E91 F0 06	BEQ BIGPRG	Only save up to 128 bytes at most
0120: 0E93 C9 0F	CMPIM \$0F	0F is code for end of program
0130: 0E95 F0 04	BEQ GOTEND	Branch if end found
0140: 0E97 D0 F5	BNE FEND	Or loop until end

```

0150: 0E99 E6 09      BIGPRG INC TO +01 Adjust FAP and TAP
0160: 0E9B 84 08      GOTEND STY TO
0170: 0E9D A2 04      LDXIM $04 This is then a copy of save
                                in monitor
0180: 0E9F B5 05      PUTADD LDAZX FROM -01
0190: 0EA1 20 B1 FE      JSR PUTBYT
0200: 0EA4 CA          DEX
0210: 0EA5 D0 F8      BNE PUTADD
0220: 0EA7 A1 06      SAVDAT LDAIX FROM
0230: 0EA9 20 B1 FE      JSR PUTBYT
0240: 0EAC 20 A0 FE      JSR COMINC
0250: 0EAF D0 F6      BNE SAVDAT
0260: 0EB1 4C 58 03      JMP AAAAA And return to entry when done
ID

```

ERRATA

NUMBER PROCESSING BUGS

Unfortunately a few errors crept into the 1st part of my article on the number processor interface for the Acorn in the May issue. The errors occurred in the section describing errors, which should read as follows.

Possible errors will occur when calculating:-

```

Ln X or LOG X when X<0
Any result X<107-99 or X>107100
TAN 90,270,450 DEG etc
SIN,COS, or TAN X when X>9000 DEG
INVERSE SIN,COS, or TAN X when |X|>1 or |X|<107-50
SQRT X when X<0
Division by zero

```

The other errors were in the table of instruction codes and were:-

a)The following codes were incorrect or missing

```

The constant PI has code 0D
y*x has code 38
/, (M/X→M) has code 3C
X↔M has code 1B
X→M has code 1C
M→X has code 1D
X↔Y has code 30
Error clear has code 2B

```

b)The conditional jumps test the following conditions and jump if true.

```

Code 11 JUMP if X=0
Code 12 JUMP if X<0
Code 13 JUMP if |X|<0
Code 14 JUMP if Error
Code 15 JUMP always

```



COMPUTER RETAILERS ASSOCIATION

1. THE COMPUTER RETAILERS ASSOCIATION.

The Computer Retailers Association is a voluntary organisation of leading micro computer dealers and service organisations. The purpose of the Association is to maintain and improve standards of trading and customer support within the industry and to act as a forum where common problems can be sorted out.

The Association is the only organisation to represent the micro computer industry in Britain. The Association has members in all parts of the country. Collectively the members of the Association have unrivalled experience of the use and application of this new technology. Of necessity Association members are familiar with the latest developments.

The members of the Association would like to put this experience at the disposal of the government and the education authorities.

2. THE PROBLEM.

Until a few years ago computers were to be found only in large organisations. The development of micro computers has changed this. For about £5,000 a small businessman can obtain a complete business micro computer system with peripherals and software for accounting, stock control and word processing.

When leased over three years such a system will cost about £170 per month, much less than the cost of a clerk or typist. Yet this modern electronic marvel can do much more than any one clerk or typist. It is reasonable to expect that over the next five years small office micro computers will become as much part of the business scene as photocopiers.

Inevitably this spread of micro computers will have a significant effect on office employment. The massive unemployment forecast by some prophets of doom is unlikely, but many routine jobs; copy typing, book keeping, filing, etc; are likely to be automated.

Many of the clerical staff affected by office automation will be re-deployed to more 'productive activities. For

example, book-keepers will have more time to spend negotiating better discounts with suppliers and speedier payment by customers. Moreover, this explosion in the country's computer population will have to be serviced by large numbers of programmers, engineers and manufacturers.

It is possible that new jobs will be created as fast as existing jobs are destroyed. However these new jobs will, on the whole, be more demanding than the ones that they replace. In the office of the future the routine work will be done by the computer whilst human beings will be required to provide the initiative, drive and decision making that no machine can give.

The key to overcoming the problems that these changes will create is education.

Every child currently going through school will have to cope with the new world of the micro computer. The schools should ensure that he or she is properly equipped to cope. In other countries particularly the USA, computer education is already an intrinsic part of the curriculum in secondary schools. This ensures that in the USA, not only are school leavers trained to work in the office of the future but also to be programmers, systems analysts and computer engineers.

3. COMPUTER SCIENCE IN SCHOOLS.

The technology of computers is changing so fast that any syllabus based upon specific equipment is bound to be out-of-date as soon as it is introduced. However, the actual techniques involved in solving problems do not change that swiftly. The emphasis in any course of instruction designed to increase a child's awareness of computing must be on the methodology of solving problems using computers.

Geography, History, Physics and Chemistry can all provide a host of practical problems that can only be effectively solved with the help of a computer. By using computers in this way a pupil can quickly relate to the methodology of problem solving and to the benefits of computing.

Writing computer programs and solving problems is enormously creative and satisfying. The positive satisfaction of successfully creating something spurs a pupil on to learn more and more about the subject and expands the ability to cope with new problems when they occur in unrelated fields. One of the problems of increasing the 'science' content of curricula is that the opportunity of creative thought is often reduced. Computing solves this problem.

Latin used to be taught in schools largely as a training for the mind in logical thought. It has not been replaced with any other subject to specifically train the mind in this fashion. Computing science is a comparable training in logical thinking.

Although a business micro computer system costs several thousands, a simple low cost computer, which will use an existing television and a standard cassette recorder and is ideally suited for use in schools, can cost less than £150. If ordered in quantity the costs would be even less.

4. SPECIFIC PROPOSALS.

We recommend the following policies to increase the effectiveness of computer science in secondary education:

- i) Computer science should be made a mainstream subject, of at least equal standing with commercial studies and mechanical drawing.
- ii) Computer science should be recognised as an entrance qualification for University. In due course it should become a required qualification for certain subjects.
- iii) The computer science course curriculum should be oriented to problem solving rather than the minutiae of particular hardware or software.
- iv) Pupils should be given as much 'hands on' experience as possible. To this end it will be necessary to equip schools with equipment. All available funds should be diverted to this end.



MICRO FOCUS

New release of CIS COBOL is faster, more portable;

Opens up new export markets

Enhancements in the latest version of CIS COBOL, Release 4.3, have improved the speed of this already highly portable compiler, Micro Focus' technical director Dr. Stewart Lang announced today. CIS COBOL is an ANSI '74 COBOL compiler and run time system developed by Micro Focus for use on microcomputers and micro-processor-based terminal systems.

The enhancements, Dr. Lang explained, have opened up new export markets by making it even simpler for Micro Focus' OEM customers to install CIS COBOL on their own equipment. The new release has already been delivered to computer manufacturers in Japan, the United States, and Europe, he added.

As well as more than doubling the speed of compilation, the latest release of CIS COBOL has improved execution speed, principally by providing mechanisms for BCD (Binary Coded Decimal) arithmetic and an optimised ISAM. Release 4.3, which has been validated by the US Federal Compiler Testing Center at Federal Low-Intermediate Level, also adds the ANSI-defined batch Debug facility.

Micro Focus has so far offered Release 4.3 only as an OEM product, for purchase by computer manufacturers. The software will be launched as an end-user product by the end of this quarter.

A number of the enhancements in this release of CIS COBOL have been achieved by the simplification of the operating system interface.

The effect of this simplification is that CIS COBOL can more easily be installed by a computer manufacturer's in-house software team, with minimal dependence on Micro Focus. This is currently being done by customers in Japan and Germany, while Micro Focus staff also carry out turn-key interfaces to order for USA and other European customers.

In addition to those already mentioned, the benefits of the enhanced interface include greater upward compatibility, for example to multi-user operating system environments, and greater modularity.

Sales of CIS COBOL to computer manufacturers worldwide contributed to Micro Focus earlier this year receiving an ICP \$1 million award; Micro Focus is the only software house in Europe to have won this award for microcomputer software.

Micro Focus also sells CIS COBOL direct to micro-computer users. Off-the-shelf object packs are currently available for any 8080, 8085 or Z80 based microcomputer running under the CP/M operating system, and for DEC LSI-II based computers running RT-II based computers running RT-III.

Release 4.3, together with all other Micro Focus products, is also available from Micro Focus' US office at 1601 Civic Center Drive, Santa Clara, CA 95050 Tel (408) 984 6961.

NEW FROM TIMEDATA...

16K BYTE + I/O FOR THE ZX80.

ZX80 Owners who have outgrown the 1K byte RAM provided in the basic machine, and who would like to expand beyond the 4K byte maximum allowed by Science of Cambridge's memory expansion unit, can now use Timedata's MZ160 dynamic RAM board to give them a full 16K bytes of program space.

The board may also be equipped to provide 24 parallel Input/Output lines — for controlling music synthesizers, model train layouts, or whatever you wish. The 24 lines may be controlled by PEEK and POKE statements or by USSR machine code routines.

Size	; 3¼" x 4½"
Ram	; 16K bytes
I/O (MZ161)	; 24 parallel bidirectional lines.
Connection	; Plugs directly into ZX80 rear connector. 23 + 23 way edge connector for I/O. Separate power connector (matching plug provided).
Power requirements	; +5v @ 200mA (may be drawn from the ZX80 supply). + 12v @ 250 mA -12v @ 20mA

MICROPROCESSOR POWER SUPPLY PZ100.

Provides three stabilised outputs, suitable for powering the ZX80 plus the MZ160 or MZ161 boards. (The PZ100's stabilised +5v output bypasses the ZX80's internal regulator, reducing heat dissipation within the ZX80's case).

Input	; 240 V AC
Outputs	; +5v @ 1A + 12v @ 300mA -12v @ 50mA

PRICES.

MZ160 memory board, built & tested, without I/O	£79.50
MZ161 memory board, built & tested, with I/O	£89.75
CN046 23 + 23 way connector (for I/O)	£3.00
MZ162 Bare PCB for MZ161, with ZX80 connector and construction details	£25.00
PZ100 Power supply, built & tested	£29.30

All prices include UK delivery and VAT @ 15%

AVAILABILITY

Commencing September 1980

MUSE RECEIVES MUCH NEEDED CASH

The department of Education and Science has today agreed to finance some of the work that MUSE (Mini and Micro computer Users in Secondary Education) has planned for the next academic year. The money will be drawn from the £9 million allocated for the development of computer education.

The final sum has yet to be determined, but we are

assured that a figure in the order of £30,000 will be available to operate the following:-

- 1) A central information service.
This will have full-time expert staffing and secretarial assistance. Many teachers using computers are in desperate need for help, advice, and information. This service will help to meet this need.
- 2) Software transfer between machines.
With a relatively small number of different makes of micro computer in schools, it seems sensible to have a team making versions of programs written for one machine, available for all.
Two research assistants will be appointed one full-time (based at Oundle) and one part-time, (based at Birmingham) to conduct this work.
(We are hoping to appoint school leavers here. Interested applicants should contact the Chairman, John Coll.)
- 3) Secretarial assistance.
This will permit the present voluntary officers of MUSE to disseminate information, good quality software, and advice based on practical experience, to a much wider audience.

MUSE is the national organisation for co-ordinating activity in Schools, Teacher Training Institutions, Colleges of Technology and other institutions with an interest in the use of computers in Secondary Education.

Meetings are held on both a regional and national basis. Courses are organised for the novice as well as the experienced user. The regional organisers provide an easily accessible local contact often able to provide immediate advice, or else able to pass questions on to other organisations or individuals with the required specialists knowledge.

Computers in Schools, the journal of MUSE, is published five times a year and it provides a forum for discussion of suitable applications, of teaching methods and the many other aspects of the use of the computer in schools.

The software library supplies programs to members at nominal rates whilst the development and standardisation of teaching packages is being actively undertaken by members. Teaching methods are a frequent topic for discussion at meetings and in 'Computers in Schools'.

Application for membership should be made to the Treasurer.

ONTEL BUYS OEM RIGHTS TO MICRO FOCUS CIS COBOL

Micro Focus' CIS COBOL software has been purchased by Ontel Corporation, the Long Island New York-based manufacturer of information systems and display computers. These systems are used in a multitude of applications including text editing, word processing and data entry.

Ontel is one of a number of computer manufacturers to have purchased OEM rights to CIS COBOL, which is a complete and highly portable software system for running COBOL programs on microcomputers and microprocessor-based terminals. Edward J. Heinze, Ontel Vice-President of Marketing, stated that CIS COBOL will provide an even

broadier scope of utilization for Ontel OEM users.

Under the terms of the contract, Ontel will sell on its equipment three Micro Focus products: Standard CIS COBOL, Compact CIS COBOL, and FORMS-2.

Standard CIS COBOL provides Ontel customers with an ANSI '74 COBOL compiler comparable to that found on most minicomputers. Late last year, CIS COBOL was validated by the US Federal Compiler Testing Center at Low-Intermediate level, the same level of certification as that of IBM's COBOL for the Series 1 and System/34, DEC's COBOL for the PDP-11, and Honeywell's COBOL for the Level 6.

Standard CIS COBOL requires a minimum of 48K bytes of user RAM and can compile and execute substantial COBOL programs on Ontel's current equipment range, which are typically supplied with 64K bytes of RAM.

For users of its earlier systems, usually equipped with 32K bytes of RAM, Ontel will supply Compact CIS COBOL. Compact CIS COBOL is a subset of the larger compiler and offers a selection of the most widely-used facilities of ANSI '74 COBOL.

The final Micro Focus product purchased by Ontel is FORMS-2, a powerful screen-editor and program generator for use with either compiler. FORMS-2 relies on the unique interactive features of CIS COBOL to permit the very rapid development of screen-processing and data entry programs.

All of this software will be offered by Ontel sales offices and distributors worldwide, the latter in the United Kingdom including Monotype Communications, Emeth Systems, Interscan Communications, and Jaserve.

Sales of CIS COBOL to computer manufacturers contributed to Micro Focus becoming in April the only European company to have received the ICP \$1 million award for microcomputer software. In addition to Ontel, CIS COBOL has been purchased by Intel, Texas Instruments, BASF, Tandberg, and QI (Europe).

Although OEM sales represent an important part of 'Micro Focus' business the company also sells CIS COBOL directly to end-users for use on many of the popular business microcomputers currently available.

Off-the-shelf CIS COBOL object packs can currently be supplied for any 8080, 8085, or Z80-based machine running the CP/M operating system or for DEC LSI-11 based computers running RT-11.

BARCLAYS BANK AND UNIVERSITY OF KENT SPONSOR NATIONAL SOFTWARE COMPETITION

Barclays Bank is now jointly sponsoring the national computer software competition run by the University of Kent at Canterbury. The 1980/81 edition will start on September 1.

The competition aims to develop an awareness among schools and colleges throughout the country of the need for industry and commerce to use computer systems. Entrants are required to design and write a computer program which has a practical application.

Mr. Brian Pearse, a Barclays general manager, said: 'We

welcome this opportunity of linking with the University of Kent on such a worthwhile project. Britain is already well known for the quality of its software and we hope the bank's involvement at this educational level will provide an added stimulus for encouraging an even higher quality in the future.'

There are two classes in the competition, one for students under 16 and another for those under 19 years of age. The winning student in each category will be presented with a Kent Software Trophy, to be held for one year, together with a £400 cash prize. In addition, the computing department of the winning student's school or college will receive up to £1000 worth of computing equipment to meet their specific needs.

Depending on the quality of entries received, a number of merit prizes ranging from £25 to £200 will also be awarded.

Closing date is February 28, 1981. Entries will be judged on imagination, usefulness and the quality of the program and its documentation. Prizes will be presented at the end of March, 1981.

Gambiet/80 scores major success in World Microcomputer Chess Championship.

Gambiet/80's achievement in coming joint third in the World Micro Chess Championship in London effectively ranks it as the best commercially available chess program in the world. The other leading entrants were either chess machines or private software developments.

Gambiet/80 was designed by Microtrend associate, Wim Rens who taught himself Z80 assembler in order to develop the program. An analytical chemist by profession, Gambiet/80 is Wim's first and only computer program; to achieve such a ranking in his first tournament is a considerable achievement. Speaking after the championship, Rens said: 'The standard was very high. With one exception, where I had an easy victory, all the games were hard-fought. Gambiet/80 is much faster than Sargon but we were still a little tight on time running as we were on a standard TRS-80. Now we know that we can beat the other programs, we now plan to give the packaged chess systems some problems.'

Work has already started on Microtrend's entry for next year's world championship.

The program is presently available on Tandy TRS-80 Level II 16K machines. Its facilities include:

- * Six levels of play from speed-chess to tournament level.
- * Graphic board display.
- * Chess clock.
- * Game record in standard notation on the screen and (optionally) on a printer.
- * Facilities to set up a board for the solution of chess problems.
- * A 'take back' facility.
- * Continual display of moves being evaluated by the program
- * Make anticipation.

Prices:

Tandy TRS-80 Level II Tape 16K	£19.95
Tandy TRS-80 Level II 16K Disk	£24.95

Versions for other computers will be available soon.

Further information from:

Jill Hebditch,
Microtrend Ltd.,
P.O. Box 51,
Pateley Bridge,
N. Yorks.
Telephone: 0423 711878

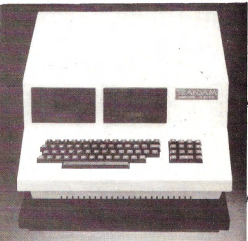
STANDARDISATION PROVES SUCCESSFUL FOR TRANSAM TUSCAN

Britain's own microcomputer manufacturer has hit on a winning combination which takes the form of the new TUSCAN S100 microcomputer. Three industry standards for the first time are incorporated on one single board computer.

Z80/A the world's most popular 8 bit micro
S100 the world's most widely used bus.

CP/M the world's most accepted disc operating system.

The heart of the TUSCAN S100 is a single board which holds Z80/A c.p.u. I/O and cassette interface, 8K user RAM, 8K Eprom plus a complete 64 x 15 character video. Five spare S100 slots make expansion easy with over 400 S100 boards available worldwide from 50 manufacturers. All this makes TUSCAN one of the most versatile single board computers available.



Due to increased volume and product rationalisation, the price of the TUSCAN kit has been reduced by over 10%. A complete kit of parts to assemble the main board is now only £235. (Less than the cost of most S100 conventional c.p.u. cards). This cost saving is reflected through the complete range of systems and a 48K RAM, twin 5¼" disc drives (double density) housed in one unit is £1481. This option is configured to run CP/M2.2 and is fully assembled and tested.

The price structure is geared to cover the whole spectrum of computing so that anyone can start on a low budget but have a versatile computer with plug in expansion. In this way the system can be configured to suit a wide variety of future applications.

PRESTEL

TRANSAM are now developing a S100 colour card with interface to PRESTEL. This will be closely allied to new applications software to complement the existing range of CP/M software.

Most CP/M software is American in origin and TRANSAM hope to further establish themselves in the U.K. software market by following up the successful launch of their first major package - TCL PASCAL. This is available to run under CP/M and was also adopted by Commodore for the PET computer. TRANSAM are developing a suite of PASCAL based application programs to be compatible with CP/M and PET computer systems. This will represent a further major standardisation in software due to the inherent portability of PASCAL which allows its implementation on several different computers which at the moment include the Superbrain, 380Z, Sorcerer and the new 80 column PET.

CP/M OPERATING SYSTEM NOW AVAILABLE FOR SHARP MZ-80K COMPUTER

Crystal Electronics, Torquay, the longest established micro-computer company in the West Country, have scored another technical triumph in producing a powerful new operating system to greatly enhance the capability of the Z 80 based Sharp MZ-80K computer.

Crystal have already produced their own BASIC for the Sharp MZ-80K computer which runs in only 9K so releasing a greater area of user memory, and have been successfully promoting sales both in the U.K. and overseas, including Japan.

Now their Crystal CP/M 2.2 operating system is available at Sharp computer dealers for around £200 excluding VAT and consists of a small, easily fitted board, operating software and instructions.

This major advance now makes the MZ-80K even more flexible and versatile than ever and improves compatibility with other machines.

The main advantage of the Crystal CP/M operating system is that it is not only low cost but that it extends the range of languages that can be used on the Sharp MZ-80K including high level languages as used on main frame computers. This means that libraries of well proven software such as the American Users Group Library can now be used, providing an almost unlimited range of applications for the machine.

Crystal Electronics have already negotiated suitable licensing arrangements for the supply of suitable software packages.

Further information is available from your local Sharp dealer or Crystal Electronics, 40 Magdalene Road, Torquay, Devon. Telephone: 0803 22699.

MUSE LAUNCHES NATIONAL INFORMATION SERVICE

MUSE is pleased to announce the launch of the National Information and Advice Centre for teachers using small computers in schools.

The Centre will be located at Birmingham, and will be manned full-time by an experienced teacher with supporting research and secretarial staff. Mr. Bob Trigger has been seconded from Marsh Hill Comprehensive School, Birmingham, and will take up the post of Information Officer from 1st January, 1981. Mr. Trigger has many years experience of teaching Computer Studies, of operating a computer-based school administration system, and more recently has been involved in a software development project for computer assisted learning.

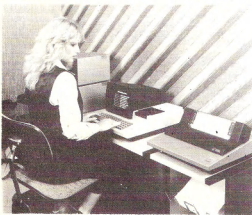
So many schools a colleges are working 'in a vacuum' in the field of micro electronics and computer education. The MUSE Information and Advice Centre will help to provide that much needed link-up for everyone from the absolute beginner to the experienced user.

The funding of the centre has been provided as part of the Government's £9 million 'Micro electronics Development Programme for Schools and Colleges'.

The Information and Advice centre will draw upon the expertise of existing MUSE members who are frequently at the forefront of developments in educational computing, in areas all over the country. There is a great wealth of experience and assistance to be tapped by coordinating this resource. Of course, anyone who feels that the Information Centre should know of their work, is encouraged to contact the main office! The address is

MUSE
FREEPOST
Bromsgrove
Worcestershire
B61 7BR

Bob Trigger's phone number is 0527 76074.



APPLICATION SOFTWARE FROM PANASONIC DISTRIBUTORS

PANASONIC Business Equipment, the UK subsidiary of Matsushita Electric, Japan's largest manufacturer of consumer electronic products, has announced a range of software now available for use with their recently introduced range of desk-top small business computers.

Available from Panasonic distributors, the choice includes ledger systems needed by all businesses – sales ledger, purchase ledger, and nominal ledger, payroll, stock control invoicing and sales statistics, and a word processing package useful in personalising direct mail letters. Additional application software specific to particular businesses is now in preparation.

The range of Panasonic computers, introduced last February, have particular appeal to small to medium sized businesses who need a simple ready-to-run computer system.

Supplied complete with three ledger packages and with a printer, a typical business system costs around £8,800.

Business applications for the new Panasonic computers are legion. Obvious applications include standard commercial systems (debtors, creditors, stock control, invoicing general ledger) to office management (word processing, reservations control, time recording, payroll), manufacturing (job costing, work in progress control, quality reporting) and retailing (sales reporting, cash register control, profit maximisation, stock control).

As an alternative to providing an accounting and management information service to a small company, the new Panasonic computers can operate as local data capture terminals for the branches of larger organisations, communicating to central mainframe systems. Each model has three inlet/outlet ports (RS232), allowing connection to communication links (or to a printer) at speeds from 110 to 9600 baud.

The Panasonic small business computers are sold in Britain by Panasonic Business Equipment (UK) Ltd, and a number of specialist companies have been appointed by Panasonic as area distributors throughout the UK. For details of local distributors and further information: Panasonic Business Equipment (UK) Ltd, 9 Connaught Street, London W2 01-262-3121

SOFTWARE FOR THE SHARP MZ-80K MICRO COMPUTER

NEWBEAR has further enhanced its large range of products for the Sharp MZ-80K with the following items:-

Chess @ £10.00 Another Newbear first, it caters for beginners grades of skill 1 to 5.

Cribbage:- £10.00 a delightful implementation and runs in 36K bytes.

Camelot:- £7.50 a dynamic game of conquering a castle with special sound effects.

Reactor:- £5.00 a test to your reaction times against a competitor.

For the home memory up grader:-

Memory Test £5.00. Essential for checking out a memory up grade.

For the machine code software programmer.

Disassembler:- £10.00 for 20K, 36K & 48K machines can be coreisident with Basic & Zen editor/assembler.

Available for callers from Newbear branches in Manchester and Birmingham and mail order from Newbury.

COBOL and Coral available on Apple II for the first time; Micro Focus offers CIS COBOL and RCC80 under CP/M

COBOL and Coral 66 – two of the most widely used professional programming languages – are now available for the Apple II personal computer for the first time. Micro Focus is offering both CIS COBOL and its Coral 66 compiler – RCC80 – on the Apple II under the popular CP/M operating system.

Standard CIS COBOL is and ANSI '74 COBOL compiler designed specifically for interactive use on a small computer systems; it supports an optional screen formatter/program generator – FORMS-2 – which greatly reduces the time taken to develop screen-processing applications.

RCC80 is a proven compiler for Coral 66, the real-time programming language which is the UK standard for defence computing. Coral 66 is also widely used in industry.

Both the Micro Focus software products run on the Apple II in conjunction with the Z80 Softcard, which makes available the Z80 instruction set in addition to the Apple's native 6502 microprocessor.

Micro Focus' chairman Brian Reynolds commented, "With over 150,000 Apple II's installed, the availability of professional languages like COBOL and Coral opens up a tremendous market opportunity to software houses and other software developers".

Micro Focus' CIS COBOL has been widely used on 8080-based and Z80-based microcomputers and is also available on DEC LSI-II equipment and the Texas Instruments T1990. It is only COBOL available on micros to have been certified, after instingent testing, by the General Services Administration of the US Government, and to have won an ICP award honouring sales in excess of 1 million.

In addition to selling individuals copies to micro-computer users, Micro Focus also licenses the software to computer manufacturers. Among manufacturers to have brought rights to CIS COBOL are Intel, Texas Instruments, BASF, CMC France, Ontel, and Q1 Europe.

RCC80, which has been certified by the UK Inter-Establishment Committee for Computer Applications (IECCA), has also been widely used for a number of years; it is also sold by GEC Semiconductors for use on Intel microcomputer development systems running under the ISIS II operating system.

OFFICE AUTOMATION: THE TRADES UNIONS' VIEW

THE introduction of new technology can stimulate new industries, create fresh opportunities and open the way to unprecedented advances. It can also create stresses, anxieties, friction and even serious industrial upheaval. The new electronic office technology is no exception.

How the new technology is introduced is crucial to its successful implementation and acceptance. Managements and Unions alike need to understand the technology, its strengths, its weaknesses and its organisational, human and industrial relations implications.

For these reasons, Urwick Nexos, as the leading British authority on the application of information technology, is holding a one-day event to assist management and others to understand Trades Unions' attitudes towards the new technology.

This event will be held on Tuesday, November 11 1980 at the Conference Centre of the Institute of Marine Engineers in London starting at 9.30am. The key speaker will be David Lea, Assistant General Secretary of the TUC, who will talk on the Union response to technological change. The event will be chaired by David Firnberg, Managing Director of Urwick Nexos.

Following David Lea, Dr. Kit Grindley of Urwick Dynamics Ltd, and Dr. Emma Bird of Urwick Nexos, will discuss the lessons learned from the introduction of computers and illustrate technology products that are available.

After lunch, two case histories will be presented. The first, on the introduction of office technology, will be presented by Douglas Broughton who is Chairman of the Bradford branch and leader and leader of the Staff Side of NALGO since 1968. The second case study concerns a new technology agreement and will be given by Robyn Dasey, a research officer of the Association of Professional, Executive, Clerical and Computer Staff (APEX).

Before the final panel discussion, Ian Benson, an official of the staff section of the Engineering Union (AUEW) will discuss the training challenge which the introduction of the new technology presents.

Further details on this important one-day event can be obtained from the Registrar, Urwick Nexos Ltd, Baylis House, Stoke Poges Lane, Slough, Berkshire SL1 3PF. The fee for the event is £120 plus VAT.

CONTROL DATA EXPANDS MINI-FLOPPY DISK FAMILY.

Control Data has announced a double-sided 5.25-inch flexible disk drive, the CDC 9409, that expands the company's family of floppy disk drives designed for original equipment manufacturers (OEM).

The CDC 9409 features full industry compatibility and unformatted data storage capacities of 218.8 thousand bytes (single density) or 437.5 thousand bytes (double density). The new drive unit is designed for use in applications such as key entry, point of sale and data collection, and with word processing, small business and personal computer systems.

Head positioning is accomplished by a band stepper mechanism that provides increased reliability, and the CDC 9409 requires no electrical adjustments or preventive maintenance during its estimated 5 year service life. Up to four drives also can be configured with a single controller for applications that require greater storage capacity.

The CDC 9409 is priced at \$225 in large quantities. Evaluation units are now available, with production deliveries currently scheduled to begin by the end of the first quarter of 1981.

NEXOS OFFICE SYSTEMS LTD. LAUNCHES THE NEXOS 2200 WORD PROCESSOR.

Nexos Office Systems Limited today launches the Nexos 2200 Word Processor, a powerful, British, stand-alone system for the 80's.

The Nexos 2200 is the first of a new generation of office products from Nexos designed to serve today's and future

users.

Mr. Muir Moffat, Managing Director of Nexos Office Systems said:- 'Our business is office systems, not merely office products. The case for buying Nexos products today rests not only on the fact that they are strong competitors in their own right, but also on the guarantee that they will key components of an integrated system four or five years hence.'

The Nexos 2200 fits attractively and unobtrusively into any office environment. The bronze screen is easy on the eyes and the adjustable screen and keyboard angles to ensure operator comfort. Light, portable and easy to operate, this new word processing system is designed to appeal to both the executive secretary and the manager.

The Nexos 2200 satisfies word processing demands from the small busy office to the largest word processing centres. Its full range of features include editing, standard layouts, standard paragraphs, automatic correspondence, term dictionary and calculator and is complimented by the real power of the latest 16-bit microprocessing technology.

Designed and built to Nexos specifications by Logica VTS Limited, the Nexos 2200 will be produced in high volume from Logica's new factory at Swindon.



Nexos provides comprehensive training after-sales support through its nationwide network of branch offices.

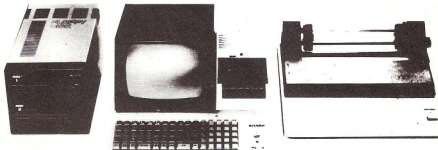
A range of furniture specially designed and manufactured by W.H. Deane is available with the Nexos 2200. The furniture, which is available in individual units, was created to complement the ergonomic lines of the 2200 and provide an overall appeal which will enhance any office.

The Nexos 2200 is available in the UK from Nexos (United Kingdom) Limited's sales offices; in Ireland and West Germany from Nexos subsidiary companies; in Holland, Belgium, Norway through the Terminal Mart Group; and in Sweden from Goran Waerner AB.

ETC ETC ETC ETC ETC
ETC ETC ETC ETC ETC
ETC ETC ETC ETC ETC

Sharp MZ-80K at Microdigital

The quality single unit computer.



SHARP

New Low prices

	Net	Vat	Total
MZ-80K Computer 48K RAM	500.00	75.00	575.00
MZ-80FD dual disk drive	780.00	117.00	897.00
MZ-80P3 printer	500.00	75.00	575.00
MZ-80U/O interface unit	84.00	12.00	96.00
MZ-80FDK extra disk drives	680.00	102.00	782.00
MZ-80T20C machine language	18.00	2.70	20.00
MZ-80TU assembler	38.00	5.70	43.70

Not a Kit

Works the same day you buy it.

Japanese

The same quality they have put into cars and Hi-Fi.

Single Unit

No trailing leads and wires

Z80

More registers and instructions than other processors

Tape Basic

You don't get left with obsolete ROMS

Tape counter

Know where you are on the tape.

Sound

Built-in music synthesiser with 3 Octaves.

Fast loading

Cassette interface runs at 1200 bps.

Other features - 79 keyboard up to 48K RAM, on screen editing, real time clock 256 different characters, 10 inch video display 80 x 50 bit mapped graphics.

The Basis of System Expansion Interface Unit MZ-801/O

The MZ-801/O interface unit connects the central processing unit (CPU) with other terminal units and makes possible further expansion of the system.

The interface unit can hold up to five different interface cards and utilizes its own built-in power source.

Fast and Legible Printing of Characters and Graphics Dot Printer MZ-80P3

By parallel data input, the MZ-80P3 prints characters on ten-inch wide paper, 80 characters to the line, at a speed of approximately 1.2 lines per second. The "tractor feed" system prevents paper slipping and produces clear print at high speed. A variety of characters can be printed by the MZ-80P3, including both upper and lower case letters, numerals and graphics.

Large Memory Capability in a Compact Unit Floppy Disc MZ-80FD

Memory capacity up to 280K bytes can be accessed quickly and easily from dual driven standard 5.25 inch floppy discs.

Specifications

MZ-801/O

Interface system

Serial

Unibus interface card

Power consumption

Power Supply

Operating temperature

Storage temperature

Dimensions

Weight

MZ-80FD

Memory capacity

No. of tracks

No. of sectors

Working conditions

Rated voltage

Power consumption

Outer dimensions

Weight

Options

Floppy interface card

Disk BASIC

For cable for connection

MZ-80P3

Printing method

Feed method

Printing capacity

Kind of printed characters

Character make-up

Size of character

Printing speed

Line/line space

Head sweep direction

Operation facilities

Interface

Print recording paper

ink ribbon

Power supply

Power consumption

Working temperature

Working humidity

Storage temperature

Storage humidity

Outer dimensions

Weight

*Specifications and design subject to change without notice

Parallel interface

TI level

Up to 5 sheets

Printer interface card, Floppy Disc interface card, Colour display interface card, Universal interface card, etc

40W

Local voltage, 50Hz

0 to 35°C

-15 to 60°C

200(W) x 320(D) x 130(H)(mm)

34kg

143K bytes/drive (286K bytes/Unit)

70

18 (per track)

4 to 25°C, 28 to 80% (relative humidity)

Local voltage, 50Hz

40W

200(W) x 320(D) x 130(H)(mm)

74kg

MZ-80U/O

MZ-80P3

MZ-80FD

MZ-80P3

Serial dot matrix method

Tractor feed method

80 characters/line

40 characters/line (Double size character display)

225 kinds excluding the space code

6 x 7 dots

12 x 7 dots (Double size character display)

Width 2.2mm Height 3.1mm

About 1.2 lines/sec (at 25°C)

2.5mm (in normal mode)

Left Right

Power supply & paper feeding

Conforming to Bandman interface

(1) 100% Parallel paper

(2) Size (Width) 102 to 254mm (4 to 10 inches)

Note: In the case of printing 80 characters per line, use paper of 254mm width. Copy possible

(1) Colour Single (Black)

(2) Size 1.38mm(W) x 11.00mm(D)

(3) 1/4" About 2 million letters

Local voltage, 50Hz

80W

5 to 40°C

10 to 80% (No dew condensation)

-20 to 50°C

5 to 85% (No dew condensation)

410(W) x 385(D) x 190(H)(mm)

10.5kg

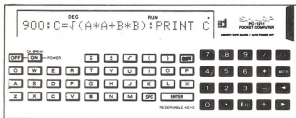


Office:
14 CASTLE STREET,
LIVERPOOL L2 0TA.
Registered in England No. 1375098

Retail Premises at:
25 BRUNSWICK STREET,
LIVERPOOL L2 0PJ.
Tel: 051-227 2535/6/7

Sharp Pocket Computer at Microdigital

A genuine advance in technology.



Adoption of Basic Language

For Programming, the PC-1211 employs the BASIC language, used widely from beginners to professionals. This simple programming method can easily be carried out by referring to the flow chart. Moreover, formulas can be entered as they are normally written. These innovative functions are designed with ease of operation in mind.

The PC-1211 also serves as an ideal "stepping stone" to professional computers.

Dot matrix display - up to 24 digits with rolling writer.

Characters as well as numerals are displayed with the dot matrix display enabling the operator, in a sense to communicate with the unit. The BASIC language can be used to its full potential. The display panel makes it possible to display portions of the program (line by line) visual instruction asking for data and showing calculation results.

Program capacity 1424 steps. 26 memories with memory safe guard.

The PC-1211 has a large memory capacity in spite of its slim, compact body. Due to the memory safe guard circuit, information in memory is maintained even after the power is turned off.

Programming is by an efficient "one-command, one-step" system. According to your needs, steps can also be used as a memory. (8 steps is equivalent to 1 memory).

Reservable key and definable key systems.

*The reservable key system makes it possible to reserve a key

for a function of command which is used frequently. It can easily be recalled by the touch of a key when putting in a formula either during manual calculation or programming.

*The definable key system defines 18 programs for each key. Whenever you need a certain program, you can recall and run it with the touch of the proper key.

Programs and data can be saved in and loaded from a tape recorder.

The cassette tape recorder can be used as an external memory device.

(Cassette interface CE-121 is optional)

By saving programs or data on a cassette tape, the information can be loaded whenever necessary. It is also possible to search the saved program data automatically by file name or load it for use during the program calculation.

Other features

- * Long-life operation, Auto power-off function.
- * Playback function enables correction by displaying the formula with a single touch of a key.
- * Effective tone function is designed to identify the program. (A beep sound can be input during programming.)

	Nett	Vat	Total
PC-1211	84.00	12.60	96.60
CE-121	12.00	1.80	13.80

Specifications SHARP POCKET COMPUTER

Model: Number of calculation: digits: Calculation system: Program system: Program language: Capacity:	PC-1211 10 digits (mantissa) + 2 digits (exponent) According to mathematical formula (with priority judging function) Stored system BASIC Data memory: Max. 1424 steps Fixed memory: 26 pcs. Flexible memory (common): 178 pcs. Reserve memory: Max. 48 steps (reserve PROGRAM, Max. 18 kinds) 80 characters Input buffer: For data: 8 stacks For function: 16 stacks (in parentheses, 15 levels) 4 stacks For subroutine: 4 stacks For FOR-NEXT statement: 4 stacks Calculations: Four arithmetic calculations, power calculation, trigonometric and inverse trigonometric functions, logarithmic and exponential functions, angular conversion, extraction of square root, sign function, absolutes, integers,	Editing functions: Cursor shifting (><) < Insertion (INS) Deletion (DEL) Line up and down () External memory: Function: Memory protection: Display: Component: Power supply: Power consumption: Operating temperature: Dimensions: Weight: Accessories:	and logical calculations. Cursor shifting (><) < Insertion (INS) Deletion (DEL) Line up and down () By using the optionally available cassette interface (CE-121), program, reserve program, and data memory can be saved or loaded to or from cassette tape recorder. CMOS battery back-up 24-digit alphanumeric dot matrix liquid crystal display. CMOS LSI, ETC. ● Alkaline manganese battery (LR-44) x 3 (built-in) Approx. 100 hours ● Silver oxide battery (G-13 or S15 type) x 3 Approx. 300 hours 4.5V ... [DC] 0.009W (with CE-121) 4.5V ... [DC] 0.011W (with CE-121) 0°C - 40°C [32°F - 104°F] 175(W) x 70(D) x 15(H)mm 5-7/8" (W) x 19/32" (H) Approx. 170g (0.37 lbs) Hard case, battery x 3 (built-in), applications manual, beginner's textbook for "BASIC", template x 2
---	---	---	--



Office:
14 CASTLE STREET,
LIVERPOOL L2 0TA
Registered in England No. 1375098

Retail Premises at:
25 BRUNSWICK STREET,
LIVERPOOL L2 0PJ
Tel: 051-227 2535/6/7